

GigaDevice Semiconductor Inc.

**GD32F30x
ARM® Cortex™-M4 32-bit MCU**

**固件库
使用指南**

1.0 版本

(2018 年 12 月)

目录

目录	2
图索引	5
表索引	6
1. 介绍	24
1.1. 文档和固件库规则	24
1.1.1. 外设缩写	24
1.1.2. 命名规则	25
2. 固件库概述	26
2.1. 文件组织结构	26
2.1.1. Examples 文件夹	27
2.1.2. Firmware 文件夹	27
2.1.3. Template 文件夹	27
2.1.4. Utilities 文件夹	30
2.2. 固件库文件描述	31
3. 外设固件库	32
3.1. 外设固件库概述	32
3.2. ADC	32
3.2.1. 外设寄存器描述	32
3.2.2. 外设库函数说明	33
3.3. BKP	69
3.3.1. 外设寄存器说明	69
3.3.2. 外设库函数说明	70
3.4. CAN	85
3.4.1. 外设寄存器说明	85
3.4.2. 外设库函数说明	86
3.5. CRC	111
3.5.1. 外设寄存器说明	111
3.5.2. 外设库函数说明	111
3.6. CTC	116
3.6.1. 外设寄存器说明	116
3.6.2. 外设库函数说明	117
3.7. DAC	133
3.7.1. 外设寄存器说明	133
3.7.2. 外设库函数说明	134

3.8. DBG	153
3.8.1. 外设寄存器说明.....	153
3.8.2. 外设库函数说明.....	153
3.9. DMA	161
3.9.1. 外设寄存器说明.....	161
3.9.2. 外设库函数说明.....	162
3.10. ENET	188
3.10.1. 外设寄存器描述.....	188
3.10.2. 外设库函数说明.....	191
3.11. EXMC.....	290
3.11.1. 外设寄存器说明.....	290
3.11.2. 外设库函数说明.....	290
3.12. EXTI.....	315
3.12.1. 外设寄存器说明.....	315
3.12.2. 外设库函数说明.....	315
3.13. FMC	324
3.13.1. 外设寄存器说明.....	324
3.13.2. 外设库函数说明.....	325
3.14. FWDGT.....	353
3.14.1. 外设寄存器说明.....	354
3.14.2. 外设库函数说明.....	354
3.15. GPIO	359
3.15.1. 外设寄存器说明.....	359
3.15.2. 外设库函数说明.....	360
3.16. I2C	378
3.16.1. 外设寄存器说明.....	378
3.16.2. 外设库函数说明.....	379
3.17. MISC	407
3.17.1. 外设寄存器说明.....	407
3.17.2. 外设库函数说明.....	409
3.18. PMU	416
3.18.1. 外设寄存器说明.....	416
3.18.2. 外设库函数说明.....	417
3.19. RCU	431
3.19.1. 外设寄存器说明.....	432
3.19.2. 外设库函数说明.....	432
3.20. RTC.....	473
3.20.1. 外设寄存器描述.....	473
3.20.2. 外设库函数描述.....	473

3.21. SDIO	483
3.21.1. 外设寄存器说明.....	483
3.21.2. 外设库函数说明.....	484
3.22. SPI	524
3.22.1. 外设寄存器说明.....	524
3.22.2. 外设库函数说明.....	524
3.23. TIMER	558
3.23.1. 外设寄存器说明.....	558
3.23.2. 外设库函数说明.....	559
3.24. USART.....	631
3.24.1. 外设寄存器说明.....	631
3.24.2. 外设库函数说明.....	632
3.25. WWDGT.....	675
3.25.1. 外设寄存器说明.....	675
3.25.2. 外设库函数说明.....	675
3.26. USBFS.....	681
4. 版本历史	682

图索引

图 2-1. GD32F30x 固件库文件组织结构	26
图 2-2. 选择外设例程文件	28
图 2-3. 拷贝外设例程文件	29
图 2-4. 打开工程文件	29
图 2-5. 配置工程文件	30
图 2-6. 编译调试下载	30

表索引

表 1-1. 外设缩写	24
表 2-1. 固件函数库文件描述	31
表 3-1. 外设固件库函数描述格式	32
表 3-2. ADC 寄存器	32
表 3-3. ADC 库函数	33
表 3-4. 函数 adc_deinit	35
表 3-5. 函数 adc_enable	35
表 3-6. 函数 adc_disable	36
表 3-7. 函数 adc_calibration_enable	37
表 3-8. 函数 adc_dma_mode_enable	37
表 3-9. 函数 adc_dma_mode_disable	38
表 3-10. 函数 adc_tempsensor_vrefint_enable	38
表 3-11. 函数 adc_tempsensor_vrefint_disable	39
表 3-12. 函数 adc_resolution_config	40
表 3-13. 函数 adc_discontinuous_mode_config	41
表 3-14. 函数 adc_mode_config	42
表 3-15. 函数 adc_special_function_config	43
表 3-16. 函数 adc_data_alignment_config	44
表 3-17. 函数 adc_channel_length_config	45
表 3-18. 函数 adc_regular_channel_config	46
表 3-19. 函数 adc_inserted_channel_config	47
表 3-20. 函数 adc_inserted_channel_offset_config	49
表 3-21. 函数 adc_external_trigger_config	50
表 3-22. 函数 adc_external_trigger_source_config	51
表 3-23. 函数 adc_software_trigger_enable	54
表 3-24. 函数 adc_regular_data_read	55
表 3-25. 函数 adc_inserted_data_read	55
表 3-26. 函数 adc_sync_mode_convert_value_read	56
表 3-27. 函数 adc_flag_get	57
表 3-28. 函数 adc_flag_clear	58
表 3-29. 函数 adc_interrupt_flag_get	59
表 3-30. 函数 adc_interrupt_flag_clear	60
表 3-31. 函数 adc_interrupt_enable	61
表 3-32. 函数 adc_interrupt_disable	61
表 3-33. 函数 adc_watchdog_single_channel_enable	62
表 3-34. 函数 adc_watchdog_group_channel_enable	63
表 3-35. 函数 adc_watchdog_disable	64
表 3-36. 函数 adc_watchdog_threshold_config	65
表 3-37. 函数 adc_oversample_mode_config	65
表 3-38. 函数 adc_oversample_mode_enable	68

表 3-39. 函数 adc_oversample_mode_disable	69
表 3-40. BKP 寄存器	69
表 3-41. BKP 库函数	70
表 3-42. 函数 bkp_deinit.....	71
表 3-43. 函数 bkp_write_data	71
表 3-44. 函数 bkp_read_data	72
表 3-45. 函数 bkp_rtc_calibration_output_enable.....	73
表 3-46. 函数 bkp_rtc_calibration_output_disable.....	73
表 3-47. 函数 bkp_rtc_signal_output_enable.....	74
表 3-48. 函数 bkp_rtc_signal_output_disable.....	75
表 3-49. 函数 bkp_rtc_output_select.....	75
表 3-50. 函数 bkp_rtc_clock_output_select	76
表 3-51. 函数 bkp_rtc_clock_calibration_direction.....	77
表 3-52. 函数 bkp_rtc_calibration_value_set	77
表 3-53. 函数 bkp_tamper_detection_enable	78
表 3-54. 函数 bkp_tamper_detection_disable	79
表 3-55. 函数 bkp_tamper_active_level_set	79
表 3-56. 函数 bkp_tamper_interrupt_enable	80
表 3-57. 函数 bkp_tamper_interrupt_disable	81
表 3-58. 函数 bkp_flag_get	81
表 3-59. 函数 bkp_flag_clear	82
表 3-60. 函数 bkp_interrupt_flag_get	83
表 3-61. 函数 bkp_interrupt_flag_clear	83
表 3-62. CAN 寄存器	85
表 3-63. CAN 库函数	86
表 3-64. 结构体 can_parameter_struct.....	87
表 3-65. 结构体 can_trasnmit_message_struct	87
表 3-66. 结构体 can_receive_message_struct	88
表 3-67. 结构体 can_filter_parameter_struct.....	88
表 3-68. 函数 can_deinit	89
表 3-69. 函数 can_struct_para_init	89
表 3-70. 函数 can_init	90
表 3-71. 函数 can_filter_init	91
表 3-72. 函数 can1_filter_start_bank	91
表 3-73. 函数 can_debug_freeze_enable	92
表 3-74. 函数 can_debug_freeze_disable	93
表 3-75. 函数 can_time_trigger_mode_enable.....	93
表 3-76. 函数 can_time_trigger_mode_disable.....	94
表 3-77. 函数 can_message_transmit	95
表 3-78. 函数 can_transmit_states	95
表 3-79. 函数 can_transmission_stop	96
表 3-80. 函数 can_message_receive	97
表 3-81. 函数 can_fifo_release	98
表 3-82. 函数 can_receive_message_length_get	99

表 3-83. 函数 can_working_mode_set	99
表 3-84. 函数 can_wakeup	100
表 3-85. 函数 can_error_get	101
表 3-86. 函数 can_receive_error_number_get	102
表 3-87. 函数 can_transmit_error_number_get	102
表 3-88. 函数 can_interrupt_enable	103
表 3-89. 函数 can_interrupt_disable	104
表 3-90. 函数 can_flag_get	105
表 3-91. 函数 can_flag_clear	107
表 3-92. 函数 can_interrupt_flag_get	108
表 3-93. 函数 can_interrupt_flag_clear	109
表 3-94. CRC 寄存器	111
表 3-95. CRC 库函数	111
表 3-96. 函数 crc_deinit	111
表 3-97. 函数 crc_data_register_reset	112
表 3-98. 函数 crc_data_register_read	113
表 3-99. 函数 crc_free_data_register_read	113
表 3-100. 函数 crc_free_data_register_write	114
表 3-101. 函数 crc_single_data_calculate	115
表 3-102. 函数 crc_block_data_calculate	115
表 3-103. CTC 寄存器	116
表 3-104. CTC 库函数	117
表 3-105. 函数 ctc_deinit	118
表 3-106. 函数 ctc_counter_enable	118
表 3-107. 函数 ctc_counter_disable	119
表 3-108. 函数 ctc_irc48m_trim_value_config	119
表 3-109. 函数 ctc_software_refsource_pulse_generate	120
表 3-110. 函数 ctc_hardware_trim_mode_config	121
表 3-111. 函数 ctc_refsource_polarity_config	121
表 3-112. 函数 ctc_refsource_signal_select	122
表 3-113. 函数 ctc_refsource_prescaler_config	123
表 3-114. 函数 ctc_clock_limit_value_config	124
表 3-115. 函数 ctc_counter_reload_value_config	125
表 3-116. 函数 ctc_counter_capture_value_read	125
表 3-117. 函数 ctc_counter_direction_read	126
表 3-118. 函数 ctc_counter_reload_value_read	127
表 3-119. 函数 ctc_irc48m_trim_value_read	127
表 3-120. 函数 ctc_interrupt_enable	128
表 3-121. 函数 ctc_interrupt_disable	129
表 3-122. 函数 ctc_interrupt_flag_get	129
表 3-123. 函数 ctc_interrupt_flag_clear	130
表 3-124. 函数 ctc_flag_get	131
表 3-125. 函数 ctc_flag_clear	132
表 3-126. DAC 寄存器	134

表 3-127. DAC 库函数	134
表 3-128. 函数 <code>dac_deinit</code>	135
表 3-129. 函数 <code>dac_enable</code>	136
表 3-130. 函数 <code>dac_disable</code>	137
表 3-131. 函数 <code>dac_dma_enable</code>	137
表 3-132. 函数 <code>dac_dma_disable</code>	138
表 3-133. 函数 <code>dac_output_buffer_enable</code>	138
表 3-134. 函数 <code>dac_output_buffer_disable</code>	139
表 3-135. 函数 <code>dac_output_value_get</code>	140
表 3-136. 函数 <code>dac_data_set</code>	140
表 3-137. 函数 <code>dac_trigger_enable</code>	141
表 3-138. 函数 <code>dac_trigger_disable</code>	142
表 3-139. 函数 <code>dac_trigger_source_config</code>	143
表 3-140. 函数 <code>dac_software_trigger_enable</code>	144
表 3-141. 函数 <code>dac_software_trigger_disable</code>	144
表 3-142. 函数 <code>dac_wave_mode_config</code>	145
表 3-143. 函数 <code>dac_wave_bit_width_config</code>	146
表 3-144. 函数 <code>dac_lfsr_noise_config</code>	147
表 3-145. 函数 <code>dac_triangle_noise_config</code>	148
表 3-146. 函数 <code>dac_concurrent_enable</code>	148
表 3-147. 函数 <code>dac_concurrent_disable</code>	149
表 3-148. 函数 <code>dac_concurrent_software_trigger_enable</code>	150
表 3-149. 函数 <code>dac_concurrent_software_trigger_disable</code>	150
表 3-150. 函数 <code>dac_concurrent_output_buffer_enable</code>	151
表 3-151. 函数 <code>dac_concurrent_output_buffer_disable</code>	151
表 3-152. 函数 <code>dac_concurrent_data_set</code>	152
表 3-153. DBG 寄存器	153
表 3-154. DBG 库函数	153
表 3-155. 枚举类型 <code>dbg_periph_enum</code>	154
表 3-156. 函数 <code>dbg_deinit</code>	155
表 3-157. 函数 <code>dbg_id_get</code>	155
表 3-158. 函数 <code>dbg_low_power_enable</code>	156
表 3-159. 函数 <code>dbg_low_power_disable</code>	157
表 3-160. 函数 <code>dbg_periph_enable</code>	158
表 3-161. 函数 <code>dbg_periph_disable</code>	158
表 3-162. 函数 <code>dbg_trace_pin_enable</code>	159
表 3-163. 函数 <code>dbg_trace_pin_disable</code>	160
表 3-164. 函数 <code>dbg_trace_pin_mode_set</code>	161
表 3-165. DMA 寄存器	162
表 3-166. DMA 库函数	162
表 3-167. 结构体 <code>dma_parameter_struct</code>	163
表 3-168. 函数 <code>dma_deinit</code>	164
表 3-169. 函数 <code>dma_struct_para_init</code>	164
表 3-170. 函数 <code>dma_init</code>	165

表 3-171. 函数 <code>dma_circulation_enable</code>	166
表 3-172. 函数 <code>dma_circulation_disable</code>	167
表 3-173. 函数 <code>dma_memory_to_memory_enable</code>	168
表 3-174. 函数 <code>dma_memory_to_memory_disable</code>	169
表 3-175. 函数 <code>dma_channel_enable</code>	169
表 3-176. 函数 <code>dma_channel_disable</code>	170
表 3-177. 函数 <code>dma_periph_address_config</code>	171
表 3-178. 函数 <code>dma_memory_address_config</code>	172
表 3-179. 函数 <code>dma_transfer_number_config</code>	173
表 3-180. 函数 <code>dma_transfer_number_get</code>	174
表 3-181. 函数 <code>dma_priority_config</code>	175
表 3-182. 函数 <code>dma_memory_width_config</code>	176
表 3-183. 函数 <code>dma_periph_width_config</code>	177
表 3-184. 函数 <code>dma_memory_increase_enable</code>	178
表 3-185. 函数 <code>dma_memory_increase_disable</code>	178
表 3-186. 函数 <code>dma_periph_increase_enable</code>	179
表 3-187. 函数 <code>dma_periph_increase_disable</code>	180
表 3-188. 函数 <code>dma_transfer_direction_config</code>	181
表 3-189. 函数 <code>dma_flag_get</code>	182
表 3-190. 函数 <code>dma_flag_clear</code>	183
表 3-191. 函数 <code>dma_interrupt_flag_get</code>	184
表 3-192. 函数 <code>dma_interrupt_flag_clear</code>	185
表 3-193. 函数 <code>dma_interrupt_enable</code>	186
表 3-194. 函数 <code>dma_interrupt_disable</code>	187
表 3-195. ENET 寄存器	188
表 3-196. ENET 库函数	191
表 3-197. 结构体 <code>enet_initpara_struct</code>	196
表 3-198. 结构体 <code>enet_descriptors_struct</code>	196
表 3-199. 结构体 <code>enet_ptp_systime_struct</code>	197
表 3-200. 函数 <code>enet_deinit</code>	197
表 3-201. 函数 <code>enet_initpara_config</code>	198
表 3-202. 函数 <code>enet_init</code>	202
表 3-203. 函数 <code>enet_software_reset</code>	204
表 3-204. 函数 <code>enet_rxframe_size_get</code>	204
表 3-205. 函数 <code>enet_descriptors_chain_init</code>	205
表 3-206. 函数 <code>enet_descriptors_ring_init</code>	206
表 3-207. 函数 <code>enet_frame_receive</code>	206
表 3-208. 函数 <code>enet_frame_transmit</code>	207
表 3-209. 函数 <code>enet_transmit_checksum_config</code>	208
表 3-210. 函数 <code>enet_enable</code>	209
表 3-211. 函数 <code>enet_disable</code>	209
表 3-212. 函数 <code>enet_mac_address_set</code>	210
表 3-213. 函数 <code>enet_mac_address_get</code>	211
表 3-214. 函数 <code>enet_flag_get</code>	212

表 3-215. 函数 enet_flag_clear	215
表 3-216. 函数 enet_interrupt_enable	216
表 3-217. 函数 enet_interrupt_disable	218
表 3-218. 函数 enet_interrupt_flag_get	220
表 3-219. 函数 enet_interrupt_flag_clear	223
表 3-220. 函数 enet_tx_enable	224
表 3-221. 函数 enet_tx_disable	225
表 3-222. 函数 enet_rx_enable	225
表 3-223. 函数 enet_rx_disable	226
表 3-224. 函数 enet_registers_get	227
表 3-225. 函数 enet_debug_status_get	227
表 3-226. 函数 enet_address_filter_enable	229
表 3-227. 函数 enet_address_filter_disable	230
表 3-228. 函数 enet_address_filter_config	230
表 3-229. 函数 enet_phy_config	232
表 3-230. 函数 enet_phy_write_read	232
表 3-231. 函数 enet_phyloopback_enable	233
表 3-232. 函数 enet_phyloopback_disable	234
表 3-233. 函数 enet_forward_feature_enable	235
表 3-234. 函数 enet_forward_feature_disable	236
表 3-235. 函数 enet_filter_feature_enable	236
表 3-236. 函数 enet_filter_feature_disable	237
表 3-237. 函数 enet_pauseframe_generate	238
表 3-238. 函数 enet_pauseframe_detect_config	239
表 3-239. 函数 enet_pauseframe_config	240
表 3-240. 函数 enet_flowcontrol_threshold_config	241
表 3-241. 函数 enet_flowcontrol_feature_enable	243
表 3-242. 函数 enet_flowcontrol_feature_disable	244
表 3-243. 函数 enet_dmaprocess_state_get	244
表 3-244. 函数 enet_dmaprocess_resume	245
表 3-245. 函数 enet_rxprocess_check_recovery	246
表 3-246. 函数 enet_txfifo_flush	247
表 3-247. 函数 enet_current_desc_address_get	247
表 3-248. 函数 enet_desc_information_get	248
表 3-249. 函数 enet_missed_frame_counter_get	249
表 3-250. 函数 enet_desc_flag_get	250
表 3-251. 函数 enet_desc_flag_set	253
表 3-252. 函数 enet_desc_flag_clear	255
表 3-253. 函数 enet_rx_desc_immediate_receive_complete_interrupt	256
表 3-254. 函数 enet_rx_desc_delay_receive_complete_interrupt	257
表 3-255. 函数 enet_rxframe_drop	257
表 3-256. 函数 enet_dma_feature_enable	258
表 3-257. 函数 enet_dma_feature_disable	259
表 3-258. 函数 enet_rx_desc_enhanced_status_get	259

表 3-259. 函数 enet_desc_select_enhanced_mode	261
表 3-260. 函数 enet_ptp_enhanced_descriptors_chain_init.....	261
表 3-261. 函数 enet_ptp_enhanced_descriptors_ring_init	262
表 3-262. 函数 enet_ptpframe_receive_enhanced_mode.....	263
表 3-263. 函数 enet_ptpframe_transmit_enhanced_mode.....	264
表 3-264. 函数 enet_desc_select_normal_mode	264
表 3-265. 函数 enet_ptp_normal_descriptors_chain_init.....	265
表 3-266. 函数 enet_ptp_normal_descriptors_ring_init	266
表 3-267. 函数 enet_ptpframe_receive_normal_mode	267
表 3-268. 函数 enet_ptpframe_transmit_normal_mode.....	267
表 3-269. 函数 enet_wum_filter_register_pointer_reset.....	268
表 3-270. 函数 enet_wum_filter_config.....	269
表 3-271. 函数 enet_wum_feature_enable.....	270
表 3-272. 函数 enet_wum_feature_disable.....	270
表 3-273. 函数 enet_msc_counters_reset	271
表 3-274. 函数 enet_msc_feature_enable.....	272
表 3-275. 函数 enet_msc_feature_disable.....	273
表 3-276. 函数 enet_msc_counters_preset_config	273
表 3-277. 函数 enet_msc_counters_get.....	274
表 3-278. 函数 enet_ptp_subsecond_2_nanosecond	275
表 3-279. 函数 enet_ptp_nanosecond_2_subsecond	276
表 3-280. 函数 enet_ptp_feature_enable	276
表 3-281. 函数 enet_ptp_feature_disable	278
表 3-282. 函数 enet_ptp_timestamp_function_config	279
表 3-283. 函数 enet_ptp_subsecond_increment_config	280
表 3-284. 函数 enet_ptp_timestamp_addend_config.....	281
表 3-285. 函数 enet_ptp_timestamp_update_config.....	282
表 3-286. 函数 enet_ptp_expected_time_config.....	283
表 3-287. 函数 enet_ptp_system_time_get.....	283
表 3-288. 函数 enet_ptp_pps_output_frequency_config.....	284
表 3-289. 函数 enet_ptp_start.....	285
表 3-290. 函数 enet_ptp_finecorrection_adjfreq.....	286
表 3-291. 函数 enet_ptp_coarsecorrection_systime_update.....	287
表 3-292. 函数 enet_ptp_finecorrection_settime	288
表 3-293. 函数 enet_ptp_flag_get	288
表 3-294. 函数 enet_initpara_reset	289
表 3-295. EXMC 寄存器.....	290
表 3-296. EXMC 库函数.....	291
表 3-297. 结构体 exmc_norsram_timing_parameter_struct.....	292
表 3-298. 结构体 exmc_norsram_parameter_struct.....	292
表 3-299. 结构体 exmc_nand_pccard_timing_parameter_struct	293
表 3-300. 结构体 exmc_nand_parameter_struct.....	293
表 3-301. 结构体 exmc_pccard_parameter_struct	293
表 3-302. 函数 exmc_norsram_deinit.....	294

表 3-303. 函数 exmc_norsram_struct_para_init.....	295
表 3-304. 函数 exmc_norsram_init	295
表 3-305. 函数 exmc_norsram_enable.....	297
表 3-306. 函数 exmc_norsram_disable.....	298
表 3-307. 函数 exmc_nand_deinit.....	298
表 3-308. 函数 exmc_nand_struct_para_init.....	299
表 3-309. 函数 exmc_nand_init	300
表 3-310. 函数 exmc_nand_enable.....	301
表 3-311. 函数 exmc_nand_disable	301
表 3-312. 函数 exmc_pccard_deinit.....	302
表 3-313. 函数 exmc_pccard_struct_para_init.....	303
表 3-314. 函数 exmc_pccard_init.....	303
表 3-315. 函数 exmc_pccard_enable.....	304
表 3-316. 函数 exmc_pccard_disable.....	305
表 3-317. 函数 exmc_norsram_page_size_config.....	306
表 3-318. 函数 exmc_nand_ecc_config	307
表 3-319. 函数 exmc_ecc_get.....	307
表 3-320. 函数 exmc_interrupt_enable.....	308
表 3-321. 函数 exmc_interrupt_disable.....	309
表 3-322. 函数 exmc_flag_get	310
表 3-323. 函数 exmc_flag_clear	311
表 3-324. 函数 exmc_interrupt_flag_get	313
表 3-325. 函数 exmc_interrupt_flag_clear	314
表 3-326. EXTI 寄存器	315
表 3-327. EXTI 库函数	315
表 3-328. 函数 exti_deinit	316
表 3-329. 函数 exti_init	317
表 3-330. 函数 exti_interrupt_enable	318
表 3-331. 函数 exti_interrupt_disable	318
表 3-332. 函数 exti_event_enable	319
表 3-333. 函数 exti_event_disable	320
表 3-334. 函数 exti_software_interrupt_enable	320
表 3-335. 函数 exti_software_interrupt_disable	321
表 3-336. 函数 exti_flag_get	321
表 3-337. 函数 exti_flag_clear	322
表 3-338. 函数 exti_interrupt_flag_get	323
表 3-339. 函数 exti_interrupt_flag_clear	323
表 3-340. FMC 寄存器	324
表 3-341. FMC 固件库函数	325
表 3-342. fmc_state_enum	326
表 3-343. fmc_int_enum	327
表 3-344. fmc_flag_enum	327
表 3-345. fmc_interrupt_flag_enum	328
表 3-346. 函数 fmc_wscnt_set.....	328

表 3-347. 函数 fmc_unlock.....	329
表 3-348. 函数 fmc_bank0_unlock.....	329
表 3-349. 函数 fmc_bank1_unlock.....	330
表 3-350. 函数 fmc_lock.....	331
表 3-351. 函数 fmc_bank0_lock	331
表 3-352. 函数 fmc_bank1_lock	332
表 3-353. 函数 fmc_page_erase	332
表 3-354. 函数 fmc_mass_erase	333
表 3-355. 函数 fmc_bank0_erase	334
表 3-356. 函数 fmc_bank1_erase	334
表 3-357. 函数 fmc_word_program.....	335
表 3-358. 函数 fmc_halfword_program.....	336
表 3-359. 函数 fmc_word_reprogram.....	337
表 3-360. 函数 ob_unlock.....	337
表 3-361. 函数 ob_lock.....	338
表 3-362. 函数 ob_erase.....	339
表 3-363. 函数 ob_write_protection_enable	339
表 3-364. 函数 ob_security_protection_config	340
表 3-365. 函数 ob_user_write	341
表 3-366. 函数 ob_data_program.....	342
表 3-367. 函数 ob_user_get.....	343
表 3-368. 函数 ob_data_get	343
表 3-369. 函数 ob_write_protection_get	344
表 3-370. 函数 ob_spc_get.....	345
表 3-371. 函数 fmc_interrupt_enable	345
表 3-372. 函数 fmc_interrupt_disable	346
表 3-373. 函数 fmc_flag_get	347
表 3-374. 函数 fmc_flag_clear	348
表 3-375. 函数 fmc_interrupt_flag_get.....	349
表 3-376. 函数 fmc_interrupt_flag_clear.....	350
表 3-377. 函数 fmc_bank0_state_get	351
表 3-378. 函数 fmc_bank1_state_get	352
表 3-379. 函数 fmc_bank0_ready_wait.....	352
表 3-380. 函数 fmc_bank1_ready_wait.....	353
表 3-381. FWDGT 寄存器.....	354
表 3-382. FWDGT 库函数.....	354
表 3-383. 函数 fwdgt_write_enable.....	354
表 3-384. 函数 fwdgt_write_disable.....	355
表 3-385. 函数 fwdgt_enable	356
表 3-386. 函数 fwdgt_counter_reload	356
表 3-387. 函数 fwdgt_config	357
表 3-388. 函数 fwdgt_flag_get	358
表 3-389. GPIO 寄存器.....	359
表 3-390. GPIO 库函数.....	360

表 3-391. 函数 gpio_deinit	360
表 3-392. 函数 gpio_afio_deinit	361
表 3-393. 函数 gpio_init	362
表 3-394. 函数 gpio_bit_set	363
表 3-395. 函数 gpio_bit_reset	364
表 3-396. 函数 gpio_bit_write	365
表 3-397. 函数 gpio_port_write	366
表 3-398. 函数 gpio_input_bit_get	367
表 3-399. 函数 gpio_input_port_get	367
表 3-400. 函数 gpio_output_bit_get	368
表 3-401. 函数 gpio_output_port_get	369
表 3-402. 函数 gpio_pin_remap_config	370
表 3-403. 函数 gpio_ethernet_phy_select	372
表 3-404. 函数 gpio_exti_source_select	373
表 3-405. 函数 gpio_event_output_config	374
表 3-406. 函数 gpio_event_output_enable	375
表 3-407. 函数 gpio_event_output_disable	375
表 3-408. 函数 gpio_pin_lock	376
表 3-409. 函数 gpio_compensation_config	377
表 3-410. 函数 gpio_compensation_flag_get	378
表 3-411. I2C 寄存器	378
表 3-412. I2C 库函数	379
表 3-413. 函数 i2c_deinit	380
表 3-414. 函数 i2c_clock_config	381
表 3-415. 函数 i2c_mode_addr_config	382
表 3-416. 函数 i2c_smbus_type_config	383
表 3-417. 函数 i2c_ack_config	384
表 3-418. 函数 i2c_ackpos_config	384
表 3-419. 函数 i2c_master_addressing	385
表 3-420. 函数 i2c_dualaddr_enable	386
表 3-421. 函数 i2c_enable	387
表 3-422. 函数 i2c_disable	388
表 3-423. 函数 i2c_start_on_bus	388
表 3-424. 函数 i2c_stop_on_bus	389
表 3-425. 函数 i2c_data_transmit	389
表 3-426. 函数 i2c_data_receive	390
表 3-427. 函数 i2c_dma_enable	391
表 3-428. 函数 i2c_dma_last_transfer_enable	392
表 3-429. 函数 i2c_stretch_scl_low_config	392
表 3-430. 函数 i2c_slave_response_to_gcall_config	393
表 3-431. 函数 i2c_software_reset_config	394
表 3-432. 函数 i2c_pec_enable	395
表 3-433. 函数 i2c_pec_transfer_enable	396
表 3-434. 函数 i2c_pec_value_get	397

表 3-435. 函数 i2c_smbus_issue_alert.....	397
表 3-436. 函数 i2c_smbus_arp_enable	398
表 3-437. 函数 i2c_flag_get.....	399
表 3-438. 函数 i2c_flag_clear.....	401
表 3-439. 函数 i2c_interrupt_enable.....	402
表 3-440. 函数 i2c_interrupt_disable.....	403
表 3-441. 函数 i2c_interrupt_flag_get	404
表 3-442. 函数 i2c_interrupt_flag_clear	406
表 3-443. NVIC 寄存器	407
表 3-444. Systick 寄存器	408
表 3-445. 枚举类型 IRQn_Type	409
表 3-446. MISC 库函数	411
表 3-447. 函数 nvic_priority_group_set.....	411
表 3-448. 函数 nvic_irq_enable	412
表 3-449. 函数 nvic_irq_disable	413
表 3-450. 函数 nvic_vector_table_set	413
表 3-451. 函数 system_lowpower_set.....	414
表 3-452. 函数 system_lowpower_reset	415
表 3-453. 函数 systick_clksource_set.....	416
表 3-454. PMU 寄存器	417
表 3-455. PMU 库函数	417
表 3-456. 函数 pmu_deinit	418
表 3-457. 函数 pmu_lvd_select	418
表 3-458. 函数 pmu_ldo_output_select.....	419
表 3-459. 函数 pmu_lvd_disable	420
表 3-460. 函数 pmu_highdriver_switch_select	420
表 3-461. 函数 pmu_highdriver_mode_enable	421
表 3-462. 函数 pmu_highdriver_mode_disable	422
表 3-463. 函数 pmu_lowdriver_mode_enable	422
表 3-464. 函数 pmu_lowdriver_mode_disable	423
表 3-465. 函数 pmu_lowpower_driver_config.....	424
表 3-466. 函数 pmu_normalpower_driver_config	424
表 3-467. 函数 pmu_to_sleepmode	425
表 3-468. 函数 pmu_to_deepsleepmode.....	426
表 3-469. 函数 pmu_to_standbymode.....	427
表 3-470. 函数 pmu_backup_write_enable	427
表 3-471. 函数 pmu_backup_write_disable	428
表 3-472. 函数 pmu_wakeup_pin_enable	429
表 3-473. 函数 pmu_wakeup_pin_disable	429
表 3-474. 函数 pmu_flag_get	430
表 3-475. 函数 pmu_flag_clear	431
表 3-476. RCU 寄存器	432
表 3-477. RCU 库函数	432
表 3-478. 函数 rcu_deinit	434

表 3-479. 函数 rcu_periph_clock_enable.....	435
表 3-480. 函数 rcu_periph_clock_disable.....	436
表 3-481. 函数 rcu_periph_clock_sleep_enable	438
表 3-482. 函数 rcu_periph_clock_sleep_disable	438
表 3-483. 函数 rcu_periph_reset_enable	439
表 3-484. 函数 rcu_periph_reset_disable	440
表 3-485. 函数 rcu_bkp_reset_enable	442
表 3-486. 函数 rcu_bkp_reset_disable	442
表 3-487. 函数 rcu_system_clock_source_config.....	443
表 3-488. 函数 rcu_system_clock_source_get	444
表 3-489. 函数 rcu_ahb_clock_config	444
表 3-490. 函数 rcu_apb1_clock_config	445
表 3-491. 函数 rcu_apb2_clock_config	446
表 3-492. 函数 rcu_ckout0_config	447
表 3-493. 函数 rcu_pll_config.....	448
表 3-494. 函数 rcu_pllpresel_config.....	449
表 3-495. 函数 rcu_pdev0_config	449
表 3-496. 函数 rcu_pdev0_config	450
表 3-497. 函数 rcu_pdev1_config	451
表 3-498. 函数 rcu_pll1_config.....	451
表 3-499. 函数 rcu_pll2_config.....	452
表 3-500. 函数 rcu_adc_clock_config	453
表 3-501. 函数 rcu_usb_clock_config	454
表 3-502. 函数 rcu_rtc_clock_config	455
表 3-503. 函数 rcu_i2s1_clock_config	456
表 3-504. 函数 rcu_i2s2_clock_config	457
表 3-505. 函数 rcu_ck48m_clock_config	457
表 3-506. 函数 rcu_flag_get	458
表 3-507. 函数 rcu_all_reset_flag_clear	460
表 3-508. 函数 rcu_interrupt_flag_get	460
表 3-509. 函数 rcu_interrupt_flag_clear	461
表 3-510. 函数 rcu_interrupt_enable	462
表 3-511. 函数 rcu_interrupt_disable	463
表 3-512. 函数 rcu_lxtal_drive_capability_config.....	464
表 3-513. 函数 rcu_osc1_stab_wait.....	465
表 3-514. 函数 rcu_osc1_on	466
表 3-515. 函数 rcu_osc1_off	467
表 3-516. 函数 rcu_osc1_bypass_mode_enable	468
表 3-517. 函数 rcu_osc1_bypass_mode_disable	469
表 3-518. 函数 rcu_hxtal_clock_monitor_enable	469
表 3-519. 函数 rcu_hxtal_clock_monitor_disable	470
表 3-520. 函数 rcu_irc8m_adjust_value_set.....	471
表 3-521. 函数 rcu_deepsleep_voltage_set.....	471
表 3-522. 函数 rcu_clock_freq_get	472

表 3-523. RTC 寄存器.....	473
表 3-524. RTC 库函数.....	474
表 3-525. 函数 <code>rtc_interrupt_enable</code>	474
表 3-526. 函数 <code>rtc_interrupt_disable</code>	475
表 3-527. 函数 <code>rtc_configuration_mode_enter</code>	476
表 3-528. 函数 <code>rtc_configuration_mode_exit</code>	476
表 3-529. 函数 <code>rtc_lwoff_wait</code>	477
表 3-530. 函数 <code>rtc_register_sync_wait</code>	478
表 3-531. 函数 <code>rtc_counter_get</code>	478
表 3-532. Function <code>rtc_counter_set</code>	479
表 3-533. 函数 <code>rtc_prescaler_set</code>	480
表 3-534. 函数 <code>rtc_alarm_config</code>	480
表 3-535. 函数 <code>rtc_divider_get</code>	481
表 3-536. 函数 <code>rtc_flag_get</code>	482
表 3-537. 函数 <code>rtc_flag_clear</code>	482
表 3-538. SDIO 寄存器	483
表 3-539. SDIO 库函数	484
表 3-540. 函数 <code>sdio_deinit</code>	486
表 3-541. 函数 <code>sdio_clock_config</code>	487
表 3-542. 函数 <code>sdio_hardware_clock_enable</code>	488
表 3-543. 函数 <code>sdio_hardware_clock_disable</code>	489
表 3-544. 函数 <code>sdio_bus_mode_set</code>	489
表 3-545. 函数 <code>sdio_power_state_set</code>	490
表 3-546. 函数 <code>sdio_power_state_get</code>	491
表 3-547. 函数 <code>sdio_clock_enable</code>	491
表 3-548. 函数 <code>sdio_clock_disable</code>	492
表 3-549. 函数 <code>sdio_command_response_config</code>	492
表 3-550. 函数 <code>sdio_wait_type_set</code>	493
表 3-551. 函数 <code>sdio_csm_enable</code>	494
表 3-552. 函数 <code>sdio_csm_disable</code>	495
表 3-553. 函数 <code>sdio_command_index_get</code>	495
表 3-554. 函数 <code>sdio_response_get</code>	496
表 3-555. 函数 <code>sdio_data_config</code>	497
表 3-556. 函数 <code>sdio_data_transfer_config</code>	499
表 3-557. 函数 <code>sdio_dsm_enable</code>	500
表 3-558. 函数 <code>sdio_dsm_disable</code>	500
表 3-559. 函数 <code>sdio_data_write</code>	501
表 3-560. 函数 <code>sdio_data_read</code>	501
表 3-561. 函数 <code>sdio_data_counter_get</code>	502
表 3-562. 函数 <code>sdio_data_counter_get</code>	503
表 3-563. 函数 <code>sdio_dma_enable</code>	503
表 3-564. 函数 <code>sdio_dma_disable</code>	504
表 3-565. 函数 <code>sdio_flag_get</code>	504
表 3-566. 函数 <code>sdio_flag_clear</code>	506

表 3-567. 函数 <code>sdio_interrupt_enable</code>	508
表 3-568. 函数 <code>sdio_interrupt_disable</code>	509
表 3-569. 函数 <code>sdio_interrupt_flag_get</code>	511
表 3-570. 函数 <code>sdio_interrupt_flag_clear</code>	513
表 3-571. 函数 <code>sdio_readwait_enable</code>	515
表 3-572. 函数 <code>sdio_readwait_disable</code>	515
表 3-573. 函数 <code>sdio_stop_readwait_enable</code>	516
表 3-574. 函数 <code>sdio_stop_readwait_disable</code>	516
表 3-575. 函数 <code>sdio_readwait_type_set</code>	517
表 3-576. 函数 <code>sdio_operation_enable</code>	518
表 3-577. 函数 <code>sdio_operation_disable</code>	518
表 3-578. 函数 <code>sdio_suspend_enable</code>	519
表 3-579. 函数 <code>sdio_suspend_disable</code>	520
表 3-580. 函数 <code>sdio_ceata_command_enable</code>	520
表 3-581. 函数 <code>sdio_ceata_command_disable</code>	521
表 3-582. 函数 <code>sdio_ceata_interrupt_enable</code>	521
表 3-583. 函数 <code>sdio_ceata_interrupt_disable</code>	522
表 3-584. 函数 <code>sdio_ceata_command_completion_enable</code>	523
表 3-585. 函数 <code>sdio_ceata_command_completion_disable</code>	523
表 3-586. SPI/I2S 寄存器	524
表 3-587. SPI/I2S 库函数	525
表 3-588. 结构体 <code>spi_parameter_struct</code>	526
表 3-589. 函数 <code>spi_i2s_deinit</code>	527
表 3-590. 函数 <code>spi_struct_para_init</code>	527
表 3-591. 函数 <code>spi_init</code>	528
表 3-592. 函数 <code>spi_enable</code>	529
表 3-593. 函数 <code>spi_disable</code>	530
表 3-594. 函数 <code>i2s_init</code>	530
表 3-595. 函数 <code>i2s_psc_config</code>	532
表 3-596. 函数 <code>i2s_enable</code>	534
表 3-597. 函数 <code>i2s_disable</code>	534
表 3-598. 函数 <code>spi_nss_output_enable</code>	535
表 3-599. 函数 <code>spi_nss_output_disable</code>	536
表 3-600. 函数 <code>spi_nss_internal_high</code>	536
表 3-601. 函数 <code>spi_nss_internal_low</code>	537
表 3-602. 函数 <code>spi_dma_enable</code>	537
表 3-603. 函数 <code>spi_dma_disable</code>	538
表 3-604. 函数 <code>spi_i2s_data_frame_format_config</code>	539
表 3-605. 函数 <code>spi_i2s_data_transmit</code>	540
表 3-606. 函数 <code>spi_i2s_data_receive</code>	541
表 3-607. 函数 <code>spi_bidirectional_transfer_config</code>	541
表 3-608. 函数 <code>spi_crc_polynomial_set</code>	542
表 3-609. 函数 <code>spi_crc_polynomial_get</code>	543
表 3-610. 函数 <code>spi_crc_on</code>	544

表 3-611. 函数 spi_crc_off.....	544
表 3-612. 函数 spi_crc_next	545
表 3-613. 函数 spi_crc_get.....	546
表 3-614. 函数 spi_ti_mode_enable.....	546
表 3-615. 函数 spi_ti_mode_disable.....	547
表 3-616. 函数 spi_nssp_mode_enable ..	548
表 3-617. 函数 spi_nssp_mode_disable	548
表 3-618. 函数 qspi_enable.....	549
表 3-619. 函数 qspi_disable.....	550
表 3-620. 函数 qspi_write_enable	550
表 3-621. 函数 qspi_read_enable	551
表 3-622. 函数 qspi_io23_output_enable.....	552
表 3-623. 函数 qspi_io23_output_disable.....	552
表 3-624. 函数 spi_i2s_interrupt_enable	553
表 3-625. 函数 spi_i2s_interrupt_disable	554
表 3-626. 函数 spi_i2s_interrupt_flag_get	555
表 3-627. 函数 spi_i2s_flag_get	556
表 3-628. 函数 spi_crc_error_clear.....	557
表 3-629.TIMER 寄存器.....	558
表 3-630. TIMER 库函数.....	559
表 3-631. 结构体 timer_parameter_struct	562
表 3-632. 结构体 timer_break_parameter_struct	562
表 3-633. 结构体 timer_oc_parameter_struct.....	563
表 3-634. 结构体 timer_ic_parameter_struct.....	563
表 3-635. 函数 timer_deinit	564
表 3-636. Function timer_struct_para_init.....	564
表 3-637. 函数 timer_init	565
表 3-638. 函数 timer_enable	566
表 3-639. 函数 timer_disable	567
表 3-640. 函数 timer_auto_reload_shadow_enable	567
表 3-641. 函数 timer_auto_reload_shadow_disable	568
表 3-642. 函数 timer_update_event_enable	569
表 3-643. 函数 timer_update_event_disable	569
表 3-644. 函数 timer_counter_alignment.....	570
表 3-645. 函数 timer_counter_up_direction	571
表 3-646. 函数 timer_counter_down_direction	572
表 3-647. 函数 timer_prescaler_config	572
表 3-648. 函数 timer_repetition_value_config	573
表 3-649. 函数 timer_autoreload_value_config	574
表 3-650. 函数 timer_counter_value_config	575
表 3-651. 函数 timer_counter_read.....	575
表 3-652. 函数 timer_prescaler_read.....	576
表 3-653. 函数 timer_single_pulse_mode_config	577
表 3-654. 函数 timer_update_source_config	578

表 3-655. 函数 timer_dma_enable.....	578
表 3-656. 函数 timer_dma_disable.....	580
表 3-657. 函数 timer_channel_dma_request_source_select	581
表 3-658. 函数 timer_dma_transfer_config	582
表 3-659. 函数 timer_event_software_generate.....	584
表 3-660. 函数 timer_break_struct_para_init	585
表 3-661. 函数 timer_break_config	585
表 3-662. 函数 timer_break_enable	587
表 3-663. 函数 timer_break_disable	587
表 3-664. 函数 timer_automatic_output_enable	588
表 3-665. 函数 timer_automatic_output_disable	588
表 3-666. 函数 timer_primary_output_config.....	589
表 3-667. 函数 timer_channel_control_shadow_config	590
表 3-668. 函数 timer_channel_control_shadow_update_config	591
表 3-669. 函数 timer_channel_output_struct_para_init.....	592
表 3-670. 函数 timer_channel_output_config	592
表 3-671. 函数 timer_channel_output_mode_config	593
表 3-672. 函数 timer_channel_output_pulse_value_config	595
表 3-673. 函数 timer_channel_output_shadow_config	596
表 3-674. 函数 timer_channel_output_fast_config.....	597
表 3-675. 函数 timer_channel_output_clear_config.....	598
表 3-676. 函数 timer_channel_output_polarity_config.....	599
表 3-677. 函数 timer_channel_complementary_output_polarity_config	600
表 3-678. 函数 timer_channel_output_state_config.....	601
表 3-679. 函数 timer_channel_complementary_output_state_config	602
表 3-680. 函数 timer_channel_input_struct_para_init	603
表 3-681. 函数 timer_input_capture_config	604
表 3-682. 函数 timer_channel_input_capture_prescaler_config	605
表 3-683. 函数 timer_channel_capture_value_register_read.....	606
表 3-684. 函数 timer_input_pwm_capture_config	607
表 3-685. 函数 timer_hall_mode_config.....	608
表 3-686. 函数 timer_input_trigger_source_select	609
表 3-687. 函数 timer_master_output_trigger_source_select.....	611
表 3-688. 函数 timer_slave_mode_select	612
表 3-689. 函数 timer_master_slave_mode_config	613
表 3-690. 函数 timer_external_trigger_config	614
表 3-691. 函数 timer_quadrature_decoder_mode_config	615
表 3-692. 函数 timer_internal_clock_config	617
表 3-693. 函数 timer_internal_trigger_as_external_clock_config.....	617
表 3-694. 函数 timer_external_trigger_as_external_clock_config	618
表 3-695. 函数 timer_external_clock_mode0_config	619
表 3-696. 函数 timer_external_clock_mode1_config	621
表 3-697. 函数 timer_external_clock_mode1_disable	622
表 3-698. 函数 timer_write_chxval_register_config	623

表 3-699. 函数 timer_output_value_selection_config.....	623
表 3-700. 函数 timer_interrupt_enable.....	624
表 3-701. 函数 timer_interrupt_disable.....	625
表 3-702. 函数 timer_interrupt_flag_get.....	626
表 3-703. 函数 timer_interrupt_flag_clear	628
表 3-704. 函数 timer_flag_get.....	629
表 3-705. 函数 timer_flag_clear.....	630
表 3-706. USART 寄存器	631
表 3-707. USART 库函数	632
表 3-708. 函数 usart_deinit	634
表 3-709. 函数 usart_baudrate_set.....	635
表 3-710. 函数 usart_parity_config.....	635
表 3-711. 函数 usart_word_length_set.....	636
表 3-712. 函数 usart_stop_bit_set	637
表 3-713. 函数 usart_enable	638
表 3-714. 函数 usart_disable	639
表 3-715. 函数 usart_transmit_config	639
表 3-716. 函数 usart_receive_config	640
表 3-717. 函数 usart_data_first_config	641
表 3-718. 函数 usart_invert_config.....	642
表 3-719. 函数 usart_receiver_timeout_enable.....	643
表 3-720. 函数 usart_receiver_timeout_disable.....	644
表 3-721. 函数 usart_receiver_timeout_threshold_config	644
表 3-722. 函数 usart_data_transmit.....	645
表 3-723. 函数 usart_data_receive.....	646
表 3-724. 函数 usart_address_config.....	647
表 3-725. 函数 usart_mute_mode_enable	647
表 3-726. 函数 usart_mute_mode_disable	648
表 3-727. 函数 usart_mute_mode_wakeup_config	649
表 3-728. 函数 usart_lin_mode_enable	650
表 3-729. 函数 usart_lin_mode_disable	650
表 3-730. 函数 usart_lin_break_detection_length_config	651
表 3-731. 函数 usart_send_break	652
表 3-732. 函数 usart_halfduplex_enable	652
表 3-733. 函数 usart_halfduplex_disable	653
表 3-734. 函数 usart_synchronous_clock_enable	654
表 3-735. 函数 usart_synchronous_clock_disable	654
表 3-736. 函数 usart_synchronous_clock_config	655
表 3-737. 函数 usart_guard_time_config	656
表 3-738. 函数 usart_smartcard_mode_enable	657
表 3-739. 函数 usart_smartcard_mode_disable	658
表 3-740. 函数 usart_smartcard_mode_nack_enable	658
表 3-741. 函数 usart_smartcard_mode_nack_disable	659
表 3-742. 函数 usart_smartcard_autoretry_config	660

表 3-743. 函数 usart_block_length_config	660
表 3-744. 函数 usart_irda_mode_enable	661
表 3-745. 函数 usart_irda_mode_disable	662
表 3-746. 函数 usart_prescaler_config	663
表 3-747. 函数 usart_irda_lowpower_config	663
表 3-748. 函数 usart_hardware_flow_rts_config	664
表 3-749. 函数 usart_hardware_flow_cts_config	665
表 3-750. 函数 usart_dma_receive_config	666
表 3-751. 函数 usart_dma_transmit_config	667
表 3-752. 函数 usart_flag_get	668
表 3-753. 函数 usart_flag_clear	669
表 3-754. 函数 usart_interrupt_enable	670
表 3-755. 函数 usart_interrupt_disable	671
表 3-756. 函数 usart_interrupt_flag_get	672
表 3-757. 函数 usart_interrupt_flag_clear	674
表 3-758. WWDGT 寄存器	675
表 3-759. WWDGT 库函数	675
表 3-760. 函数 wwdgt_deinit	676
表 3-761. 函数 wwdgt_enable	676
表 3-762. 函数 wwdgt_counter_update	677
表 3-763. 函数 wwdgt_config	678
表 3-764. 函数 wwdgt_interrupt_enable	679
表 3-765. 函数 wwdgt_flag_get	679
表 3-766. 函数 wwdgt_flag_clear	680
表 4-1. 版本历史	682

1. 介绍

本手册介绍了32位基于ARM微控制器GD32F30x固件库。

该固件库是一个固件函数包，它由程序、数据结构和宏组成，包括了GD32F30x所有外设的性能特征。该固件库还包括每一个外设的驱动描述和基于评估板的固件库使用例程。通过使用本固件库，用户无需深入掌握细节，也可以轻松应用每一个外设。使用本固件库可以大大减少用户的编程时间，从而降低开发成本。

每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。可以通过调用一组通用API(application programming interface应用编程界面)来实现对外设的驱动，这些API的结构、函数名称和参数名称都进行了标准化规范。

所有的驱动源代码都符合“MISRA-C:2004”标准（例程文件符合扩充ANSI-C标准），不会受到来自开发环境差异带来的影响。仅有启动文件取决于开发环境。

因为该固件库是通用的，并且包括了所有外设的功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，用户可以直接使用之，对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库可以作为如何设置外设的一份参考资料，可以根据实际需求对其进行调整。

此份固件库使用手册的整体架构如下：

- 文档和固件库规则；
- 固件库概述；
- 外设固件库具体描述，外设固件库例程使用说明。

1.1. 文档和固件库规则

1.1.1. 外设缩写

表 1-1. 外设缩写

外设缩写	说明
ADC	模数转换器
BKP	备份寄存器
CAN	局域网控制器模块
CRC	循环冗余校验计算单元
CTC	时钟校准控制器
DAC	数模转换器
DBG	调试模块
DMA	直接存储器访问控制器
ENET	以太网
EXMC	外部存储器控制器

外设缩写	说明
EXTI	外部中断事件控制器
FMC	闪存控制器
FWDGT	独立看门狗
GPIO/APIO	通用和备用输入/输出接口
I2C	内部集成电路总线接口
MISC	嵌套中断向量列表控制器
PMU	电源管理单元
RCU	复位和时钟单元
RTC	实时时钟
SDIO	SDIO接口
SPI/I2S	串行外设接口/片上音频接口
TIMER	定时器
USART	通用同步异步收发器
WWDGT	窗口看门狗
USBFS	通用串行总线全速接口

1.1.2. 命名规则

固件库遵从以下命名规则：

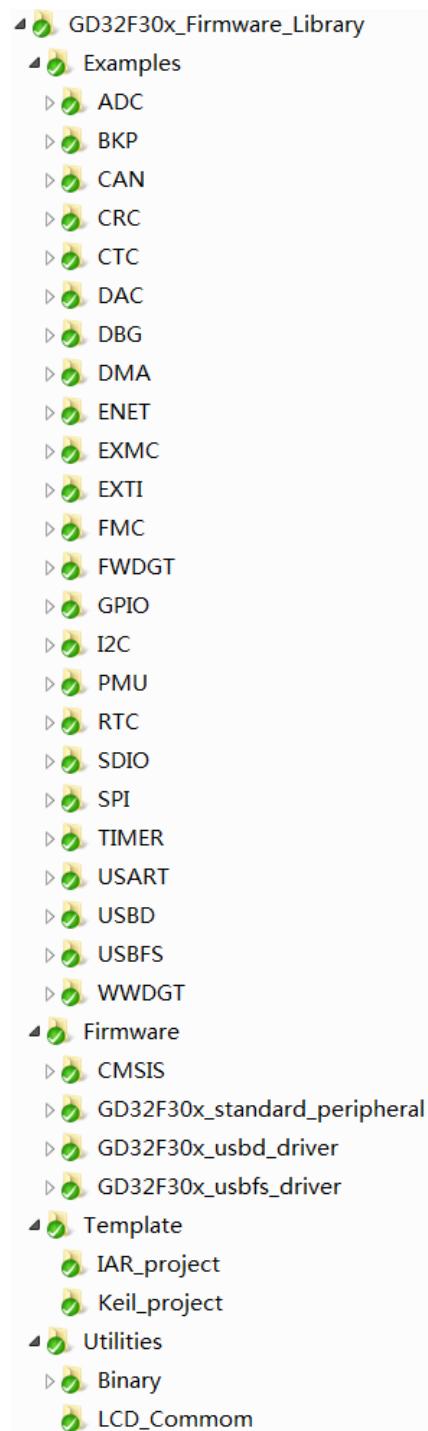
- XXX表示任一外设缩写，例如：ADC。更多缩写相关信息参阅[外设缩写](#)；
- 源文件和头文件命名都以“gd32f30x_”作为开头，例如：gd32f30x_adc.h；
- 常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。所有常量都由英文字母大写书写；
- 寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，寄存器缩写规范与本用户手册一致；
- 变量名采用全部小写，有多个单词组成的，在单词之间以下划线分隔；
- 外设函数的命名以该外设的缩写加下划线为开头，有多个单词组成的，在单词之间以下划线分隔，所有外设函数都由英文字母小写书写。

2. 固件库概述

2.1. 文件组织结构

GD32F30x_Firmware_Library, 文件组织结构见下图:

图 2-1. GD32F30x 固件库文件组织结构



2.1.1. Examples 文件夹

文件夹**Examples**, 对应每一个GD32外设均包含一个子文件夹。每个子文件夹包含了关于本外设的一个或多个例程, 来示范如何使用对应外设。每个例程子文件夹包含如下文件:

- **readme.txt**: 关于本例程的简单描述和使用说明;
- **gd32f30x_libopt.h**: 该头文件可以设置例程所使用到的外设, 由不同的“**DEFINE**”语句组成(默认情况下, 所有外设均打开);
- **gd32f30x_it.c**: 该源文件包含了所有的中断处理程序(如果未使用到中断, 则所有的函数体都为空);
- **gd32f30x_it.h**: 该头文件包含了所有的中断处理程序的原形;
- **systick.c**: 该源文件包含了使用**systick**的精准延时程序;
- **systick.h**: 该头文件包含了使用**systick**的精准延时程序的原形;
- **main.c**: 例程代码注: 所有的例程的使用, 都不受不同软件开发环境的影响。

2.1.2. Firmware 文件夹

Firmware文件夹包含组成固件库核心的所有子文件夹和文件:

- CMSIS子文件夹包含有Cortex M4内核的支持文件、基于Cortex M4内核处理器的启动代码和库引导文件以及基于GD32F30x的全局头文件和系统配置文件;
- GD32F30x_standard_peripheral子文件夹:
 - **Include**子文件夹包含了固件函数库所需的头文件, 用户无需修改该文件夹;
 - **Source**子文件夹包含了固件函数库所需的源文件, 用户无需修改该文件夹;
- GD32F30x_usbd_driver子文件夹包含了关于USBD外设的所有文件;
 - **Include**子文件夹包含了USBD外设所需的头文件, 用户无需修改该文件夹;
 - **Source**子文件夹包含了USBD外设所需的源文件, 用户无需修改该文件夹;
- GD32F30x_usbfs_driver子文件夹包含了关于USBFS外设的所有文件;
 - **Include**子文件夹包含了USBFS外设所需的头文件, 用户无需修改该文件夹;
 - **Source**子文件夹包含了USBFS外设所需的源文件, 用户无需修改该文件夹;

注: 所有代码都按照 MISRA-C:2004标准书写, 都不受不同软件开发环境的影响。

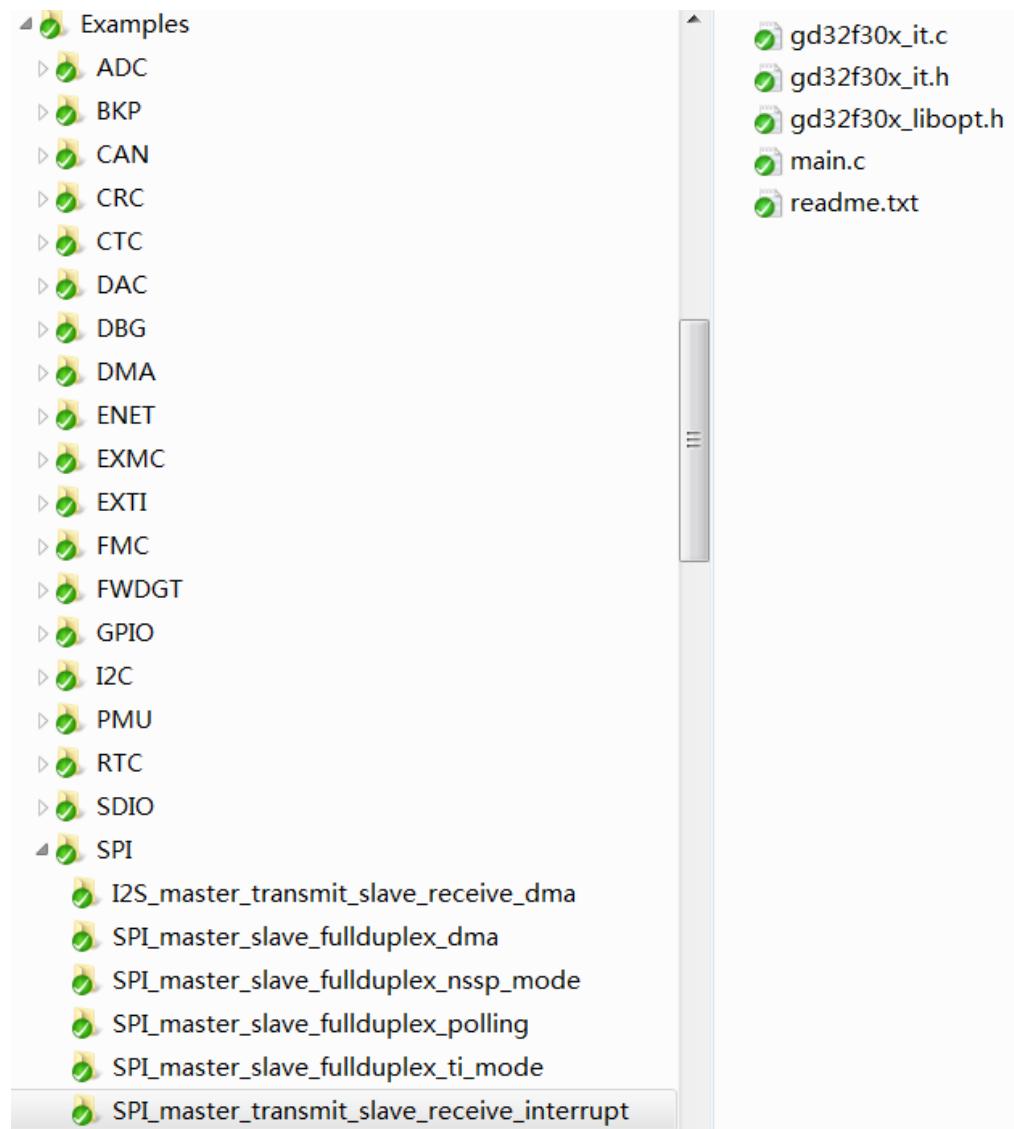
2.1.3. Template 文件夹

Template文件夹包含一个关于使用LED、USART打印、按键控制的简单例程, (IAR_project用于IAR编译环境, Keil_project用于Keil4编译环境)。用户可以使用该工程模板进行固件库例程的移植编译, 具体使用方法见下:

选择文件

打开“Examples”文件夹，选择需要测试的模块，如SPI，打开“SPI”文件夹，选择SPI的一个例程，如“SPI_master_transmit_slave_receive_interrupt”，如下图所示：

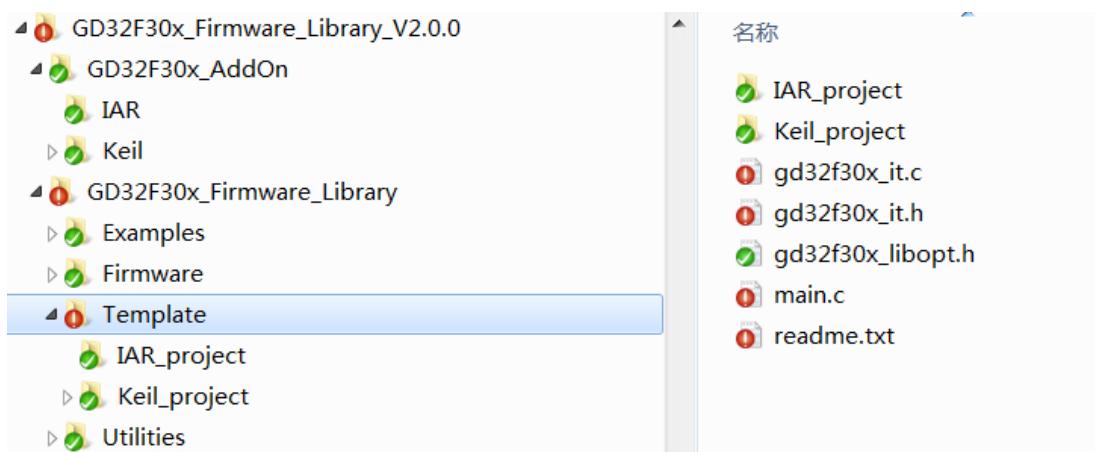
图 2-2. 选择外设例程文件



拷贝文件

打开“Template”文件夹，将“IAR_project”和“Keil_project”两个文件夹保留，其他文件都删除，然后将“SPI_master_transmit_slave_receive_interrupt”文件夹中的所有文件拷到“Template”文件夹子目录下，如下图所示：

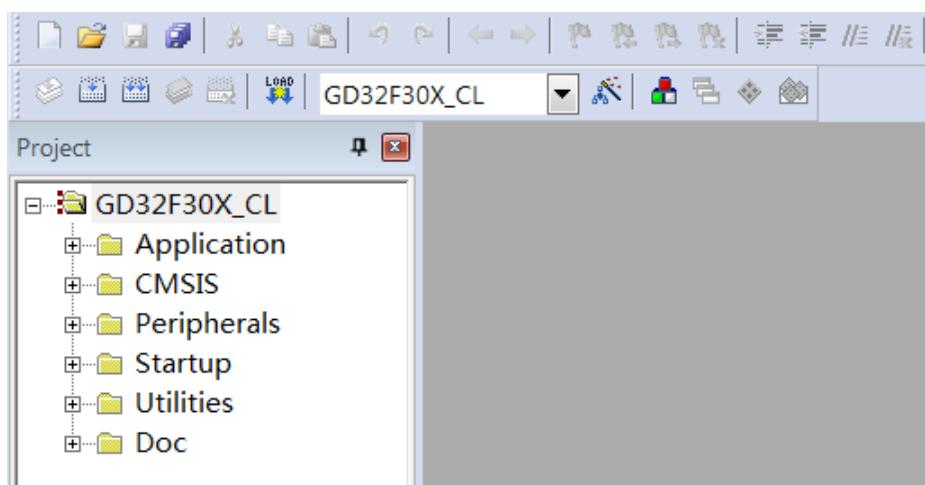
图 2-3. 拷贝外设例程文件



打开工程

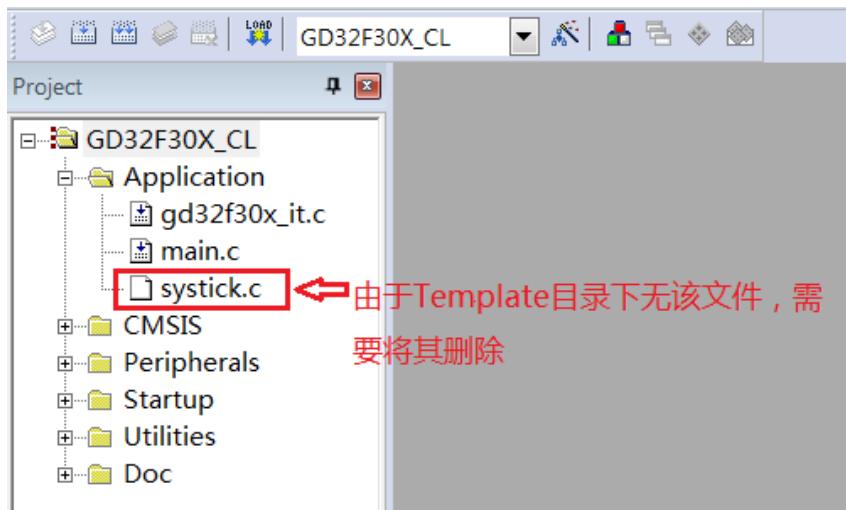
GD 提供 Keil 和 IAR 两种版本的工程，根据客户所安装的软件，打开不同的 project，如“Keil_project”，打开\Template\Keil_project\Project.uvproj，如下图所示：

图 2-4. 打开工程文件



由于不同的模块、不同的功能，会使用到不同的文件，需要根据客户选择拷贝的文件，对工程里的文件进行增加或删除，如下图所示：

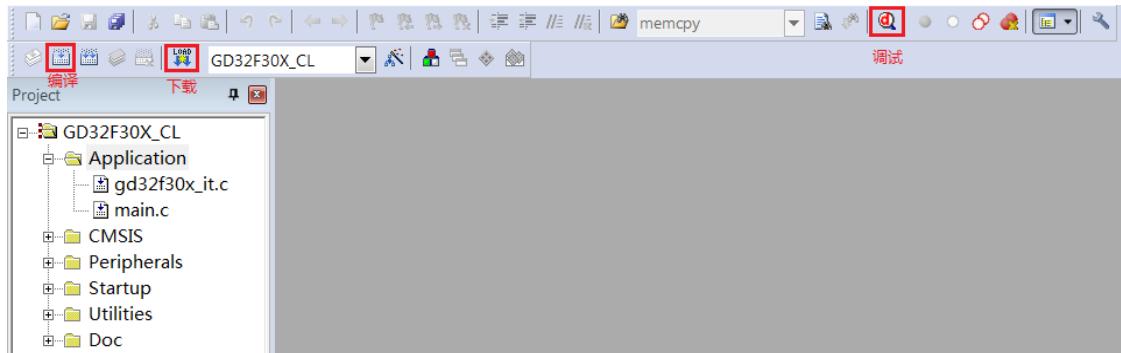
图 2-5. 配置工程文件



编译调试下载

首先编译整个工程，如果无错误，按照readme中的介绍，选择正确的跳线及连线，然后再将程序下载到目标板上，则会有如readme中描述的现象。IDE的具体使用，请参考相应的软件使用说明。如客户使用的是Keil，可见下图所示：

图 2-6. 编译调试下载



2.1.4. Utilities 文件夹

Utilities文件夹包含运行固件库例程评估板的文件：

- Binary、LCD_Common子文件夹包含有USB测试所需文件；
- gd32f30x_eval.h及gd32f307c_lcd_eval.h文件是运行固件库例程所需关于评估板的头文件；
- gd32f30x_eval.c及gd32f307c_lcd_eval.c文件是运行固件库例程所需关于评估板的源文件。

注：所有代码都按照 MISRA-C:2004标准书写，都不受不同软件开发环境的影响。

2.2. 固件库文件描述

下表列举和描述了固件库使用的主要文件。

表 2-1. 固件函数库文件描述

文件名	描述
gd32f30x_libopt.h	包含了所有外设的头文件的头文件。它是唯一一个用户需要包括在自己应用中的文件，起到应用和库之间界面的作用。
main.c	主函数体示例。
gd32f30x_it.h	头文件，包含所有中断处理函数原形。
gd32f30x_it.c	外设中断函数文件。用户可以加入自己的中断程序代码。对于指向同一个中断向量的多个不同中断请求，可以利用函数通过判断外设的中断标志位来确定准确的中断源。固件库提供了这些函数的名称。
gd32f30x_xxx.h	外设PPP的头文件。包含外设PPP函数的定义，以及这些函数使用的变量。
gd32f30x_xxx.c	由C语言编写的外设PPP的驱动源程序文件。
systick.h	systick.c的头文件。包含systick配置函数的定义，以及外部用延时函数的定义。
systick.c	systick配置与延时函数源文件。
readme.txt	固件库例程使用及配置说明文档。

3. 外设固件库

3.1. 外设固件库概述

外设固件库函数的描述格式如下表：

表 3-1. 外设固件库函数描述格式

函数名称	外设函数的名称
函数原型	原型声明
功能描述	简要解释函数是如何执行的
先决条件	调用函数前应满足的要求
被调用函数	其他被该函数调用的库函数
输入参数{in}	
XXX	输入参数描述
Xx	输入参数可选宏描述
输出参数{out}	
XXX	输出参数描述
返回值	
XXX	函数的返回值

3.2. ADC

12位ADC是一种采用逐次逼近方式的模拟数字转换器。章节[3.2.1](#)描述了ADC的寄存器列表，章节[3.2.2](#)对ADC库函数进行说明。

3.2.1. 外设寄存器描述

ADC寄存器列表如下表所示：

表 3-2. ADC 寄存器

寄存器名称	寄存器描述
ADC_STAT	状态寄存器
ADC_CTL0	控制寄存器0
ADC_CTL1	控制寄存器1

寄存器名称	寄存器描述
ADC_SAMPT0	采样时间寄存器0
ADC_SAMPT1	采样时间寄存器1
ADC_IOFFx (x=0..3)	注入通道数据偏移寄存器x
ADC_WDHT	看门狗高阈值寄存器
ADC_WDLT	看门狗低阈值寄存器
ADC_RSQ0	规则序列寄存器0
ADC_RSQ1	规则序列寄存器1
ADC_RSQ2	规则序列寄存器2
ADC_ISQ	注入序列寄存器
ADC_IDATAx(x=0..3)	注入数据寄存器x
ADC_RDATA	规则数据寄存器
ADC_OVSAMPCTL	过采样控制寄存器

3.2.2. 外设库函数说明

ADC库函数列表如下表所示：

表 3-3. ADC 库函数

库函数名称	库函数描述
adc_deinit	复位ADCx外设
adc_enable	使能ADCx外设
adc_disable	禁能ADCx外设
adc_calibration_enable	ADCx校准复位
adc_dma_mode_enable	ADCx DMA请求使能
adc_dma_mode_disable	ADCx DMA请求禁能
adc_tempsensor_vrefint_enable	温度传感器和Vrefint通道使能
adc_tempsensor_vrefint_disable	温度传感器和Vrefint通道禁能
adc_resolution_config	配置ADCx分辨率
adc_discontinuous_mode_config	配置ADC间断模式

库函数名称	库函数描述
adc_mode_config	配置ADC同步模式
adc_special_function_config	使能或禁能ADC特殊功能
adc_data_alignment_config	配置ADC数据对齐方式
adc_channel_length_config	配置规则通道组或注入通道组的长度
adc_regular_channel_config	配置ADC规则通道组
adc_inserted_channel_config	配置ADC注入通道组
adc_inserted_channel_offset_config	配置ADC注入通道组数据偏移值
adc_external_trigger_config	配置ADC外部触发
adc_external_trigger_source_config	配置ADC外部触发源
adc_software_trigger_enable	ADC软件触发使能
adc_regular_data_read	读ADC规则组数据寄存器
adc_inserted_data_read	读ADC注入组数据寄存器
adc_sync_mode_convert_value_read	在同步模式下，读ADC0和ADC1最近的一次转换结果
adc_flag_get	获取ADC标志位
adc_flag_clear	清除ADC标志位
adc_interrupt_flag_get	获取ADC中断标志位
adc_interrupt_flag_clear	清除ADC中断标志位
adc_interrupt_enable	ADC中断使能
adc_interrupt_disable	ADC中断禁能
adc_watchdog_single_channel_enable	配置ADC模拟看门狗单通道有效
adc_watchdog_group_channel_enable	配置ADC模拟看门狗在通道组有效
adc_watchdog_disable	ADC模拟看门狗禁能
adc_watchdog_threshold_config	配置ADC模拟看门狗阈值
adc_oversample_mode_config	配置ADCx过采样模式
adc_oversample_mode_enable	使能ADCx过采样
adc_oversample_mode_disable	禁能ADCx过采样

函数 adc_deinit

函数adc_deinit描述见下表:

表 3-4. 函数 adc_deinit

函数名称	adc_deinit
函数原形	void adc_deinit(uint32_t adc_periph);
功能描述	复位ADCx外设
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset ADC0 */
adc_deinit(ADC0);
```

函数 adc_enable

函数adc_enable描述见下表:

表 3-5. 函数 adc_enable

函数名称	adc_enable
函数原形	void adc_enable(uint32_t adc_periph);
功能描述	使能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设

$ADCx(x=0..2)$	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 */
adc_enable(ADC0);
```

函数 **adc_disable**

函数adc_disable描述见下表：

表 3-6. 函数 **adc_disable**

函数名称	adc_disable
函数原形	void adc_disable(uint32_t adc_periph);
功能描述	禁能ADCx外设
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
$ADCx(x=0..2)$	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 */
adc_disable(ADC0);
```

函数 **adc_calibration_enable**

函数adc_calibration_enable描述见下表：

表 3-7. 函数 adc_calibration_enable

函数名称	adc_calibration_enable
函数原形	void adc_calibration_enable(uint32_t adc_periph);
功能描述	ADCx校准复位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* ADC0 calibration and reset calibration */
adc_calibration_enable(ADC0);
```

函数 adc_dma_mode_enable

函数adc_dma_mode_enable描述见下表：

表 3-8. 函数 adc_dma_mode_enable

函数名称	adc_dma_mode_enable
函数原形	void adc_dma_mode_enable(uint32_t adc_periph);
功能描述	ADCx DMA请求使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable ADC0 DMA request */
adc_dma_mode_enable(ADC0);
```

函数 **adc_dma_mode_disable**

函数adc_dma_mode_disable描述见下表：

表 3-9. 函数 **adc_dma_mode_disable**

函数名称	adc_dma_mode_disable
函数原形	void adc_dma_mode_disable(uint32_t adc_periph);
功能描述	ADCx DMA请求禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 DMA request */
adc_dma_mode_disable(ADC0);
```

函数 **adc_tempsensor_vrefint_enable**

函数adc_tempsensor_vrefint_enable描述见下表：

表 3-10. 函数 **adc_tempsensor_vrefint_enable**

函数名称	adc_tempsensor_vrefint_enable
------	-------------------------------

函数原形	void adc_tempsensor_vrefint_enable(void);
功能描述	温度传感器和Vrefint通道使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the temperature sensor and Vrefint channel */

adc_tempsensor_vrefint_enable();
```

函数 **adc_tempsensor_vrefint_disable**

函数adc_tempsensor_vrefint_disable描述见下表：

表 3-11. 函数 **adc_tempsensor_vrefint_disable**

函数名称	adc_tempsensor_vrefint_disable
函数原形	void adc_tempsensor_vrefint_disable(void);
功能描述	温度传感器和Vrefint通道禁能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the temperature sensor and Vrefint channel */
```

```
adc_tempsensor_vrefint_disable();
```

函数 adc_resolution_config

函数adc_resolution_config描述见下表:

表 3-12. 函数 adc_resolution_config

函数名称	adc_resolution_config
函数原形	void adc_resolution_config(uint32_t adc_periph , uint32_t resolution);
功能描述	配置ADCx分辨率
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
resolution	ADC分辨率
ADC_RESOLUTION_12B	12位分辨率
ADC_RESOLUTION_10B	10位分辨率
ADC_RESOLUTION_8B	8位分辨率
ADC_RESOLUTION_6B	6位分辨率
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

函数 adc_discontinuous_mode_config

函数adc_discontinuous_mode_config描述见下表：

表 3-13. 函数 adc_discontinuous_mode_config

函数名称	adc_discontinuous_mode_config
函数原形	void adc_discontinuous_mode_config(uint32_t adc_periph, uint8_t adc_channel_group, uint8_t length);
功能描述	配置ADC间断模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
ADC_CHANNEL_DISABLE	规则通道组和注入通道组间断模式禁能
输入参数{in}	
length	间断模式下的转换数目，规则通道组取值为1..8，注入通道组取值无意义
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 discontinuous mode */
adc_discontinuous_mode_config(ADC0, ADC_REGULAR_CHANNEL, 6);
```

函数 adc_mode_config

函数adc_mode_config描述见下表：

表 3-14. 函数 adc_mode_config

函数名称	adc_mode_config
函数原形	void adc_mode_config(uint32_t mode);
功能描述	配置ADC同步模式
先决条件	-
被调用函数	-
输入参数{in}	
mode	ADC 运行模式
ADC_MODE_FREE	所有ADC运行于独立模式
ADC_DAUL_REGU_LAL_PARALLEL_IN_SERTED_PARALLEL_L	ADC0和ADC1运行在规则并行+注入并行组合模式
ADC_DAUL_REGU_LAL_PARALLEL_IN_SERTED_ROTATION_N	ADC0和ADC1运行在规则并行+交替触发组合模式
ADC_DAUL_INSERTED_PARALLEL_REGULAL_FOLLOW_UP_FAST	ADC0和ADC1运行在注入并行+快速交叉组合模式
ADC_DAUL_INSERTED_PARALLEL_REGULAL_FOLLOW_UP_SLOW	ADC0和ADC1运行在注入并行+慢速交叉组合模式
ADC_DAUL_INSERTED_PARALLEL	ADC0和ADC1运行在注入并行模式
ADC_DAUL_REGU_LAL_PARALLEL	ADC0和ADC1运行在规则并行模式
ADC_DAUL_REGU_LAL_FOLLOWUP_FAST	ADC0和ADC1运行在快速交叉模式

<i>ADC_DAUL_REGU_LAL_FOLLOWUP_SLOW</i>	ADC0和ADC1运行在慢速交叉模式
<i>ADC_DAUL_INSERTED_TRIGGER_ROTATION</i>	ADC0和ADC1运行在交替触发模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC sync mode */
adc_mode_config(ADC_MODE_FREE);
```

函数 **adc_special_function_config**

函数adc_special_function_config描述见下表：

表 3-15. 函数 **adc_special_function_config**

函数名称	adc_special_function_config
函数原形	void adc_special_function_config(uint32_t adc_periph, uint32_t function, ControlStatus newvalue);
功能描述	使能或禁能ADC特殊功能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
function	功能配置
<i>ADC_SCAN_MODE</i>	扫描模式选择
<i>ADC_INSERTED_CHANNEL_AUTO</i>	注入组自动转换

<i>ADC_CONTINUOUS_MODE</i>	连续模式选择
输入参数{in}	
newvalue	功能使能禁能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 scan mode */

adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
```

函数 **adc_data_alignment_config**

函数adc_alignment_config描述见下表：

表 3-16. 函数 **adc_data_alignment_config**

函数名称	adc_data_alignment_config
函数原形	void adc_data_alignment_config(uint32_t adc_periph, uint32_t data_alignment);
功能描述	配置ADCx数据对齐方式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
data_alignment	数据对齐方式选择
<i>ADC_DATAALIGN_RIGHT</i>	LSB 对齐

<i>ADC_DATAALIGN_LEFT</i>	MSB 对齐
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 data alignment */
adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);
```

函数 **adc_channel_length_config**

函数adc_channel_length_config描述见下表：

表 3-17. 函数 **adc_channel_length_config**

函数名称	adc_channel_length_config
函数原形	void adc_channel_length_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t length);
功能描述	配置规则通道组或注入通道组的长度
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组
输入参数{in}	
length	通道长度，规则通道组为1-16，注入通道组为1-4

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the length of ADC0 regular channel */
adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, 4);
```

函数 **adc_regular_channel_config**

函数adc_regular_channel_config描述见下表：

表 3-18. 函数 **adc_regular_channel_config**

函数名称	adc_regular_channel_config
函数原形	void adc_regular_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC规则通道组
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
rank	规则组通道序列，取值范围为0~15
输入参数{in}	
adc_channel	ADC通道选择
ADC_CHANNEL_x(x=0..17)	ADC 通道x (x=0..17)(只有ADC0, 可取值x=16和17)
输入参数{in}	
sample_time	采样时间
ADC_SAMPLETIME_1POINT5	1.5 周期

<i>ADC_SAMPLETIME_7POINT5</i>	7.5 周期
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 周期
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 周期
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 周期
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 周期
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 周期
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 regular channel */
adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);
```

函数 **adc_inserted_channel_config**

函数adc_inserted_channel_config描述见下表：

表 3-19. 函数 **adc_inserted_channel_config**

函数名称	adc_inserted_channel_config
函数原形	void adc_inserted_channel_config(uint32_t adc_periph, uint8_t rank, uint8_t adc_channel, uint32_t sample_time);
功能描述	配置ADC注入通道组
先决条件	-
被调用函数	-
输入参数{in}	

adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
rank	注入组通道序列，取值范围为0~3
输入参数{in}	
adc_channel	ADC通道选择
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC 通道x (x=0..17)(只有ADC0, 可取值x=16和17)
输入参数{in}	
sample_time	采样时间
<i>ADC_SAMPLETIME_1POINT5</i>	1.5 周期
<i>ADC_SAMPLETIME_7POINT5</i>	7.5 周期
<i>ADC_SAMPLETIME_13POINT5</i>	13.5 周期
<i>ADC_SAMPLETIME_28POINT5</i>	28.5 周期
<i>ADC_SAMPLETIME_41POINT5</i>	41.5 周期
<i>ADC_SAMPLETIME_55POINT5</i>	55.5 周期
<i>ADC_SAMPLETIME_71POINT5</i>	71.5 周期
<i>ADC_SAMPLETIME_239POINT5</i>	239.5 周期
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 inserted channel */
```

 adc_inserted_channel_config(ADC0, 1, ADC_CHANNEL_0, ADC_SAMPLETIME_7POINT5);

函数 **adc_inserted_channel_offset_config**

函数adc_inserted_channel_offset_config描述见下表:

表 3-20. 函数 adc_inserted_channel_offset_config

函数名称	adc_inserted_channel_offset_config
函数原形	void adc_inserted_channel_offset_config(uint32_t adc_periph, uint8_t inserted_channel, uint16_t offset);
功能描述	配置ADC注入通道组数据偏移值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道, x=0,1,2,3
输入参数{in}	
offset	数据偏移值, 取值范围为0~4095
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 inserted channel offset */
adc_inserted_channel_offset_config(ADC0, ADC_INSERTED_CHANNEL_0, 100);
```

函数 **adc_external_trigger_config**

函数adc_external_trigger_config描述见下表:

表 3-21. 函数 adc_external_trigger_config

函数名称	adc_external_trigger_config
函数原形	void adc_external_trigger_config(uint32_t adc_periph, uint8_t adc_channel_group, ControlStatus newvalue);
功能描述	配置ADC外部触发
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
newvalue	通道使能禁能
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 inserted channel group external trigger */
adc_external_trigger_config(ADC0, ADC_INSERTED_CHANNEL_0, ENABLE);
```

函数 adc_external_trigger_source_config

函数adc_external_trigger_source_config描述见下表：

表 3-22. 函数 adc_external_trigger_source_config

函数名称	adc_external_trigger_source_config
函数原形	void adc_external_trigger_source_config(uint32_t adc_periph, uint8_t adc_channel_group, uint32_t external_trigger_source);
功能描述	配置ADC外部触发源
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输入参数{in}	
external_trigger_source	规则通道组或注入通道组触发源
ADC0_1_EXTTRIG_REGULAR_T0_CH0	TIMER0 CH0事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T0_CH1	TIMER0 CH1事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T0_CH2	TIMER0 CH2事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T1_CH1	TIMER1 CH1事件（规则组）
ADC0_1_EXTTRIG_REGULAR_T2_CH0	TIMER2 TRGO事件（规则组）

<code>_REGULAR_T2_TR GO</code>	
<code>ADC0_1_EXTTRIG _REGULAR_T3_CH 3</code>	TIMER3 CH3事件（规则组）
<code>ADC0_1_EXTTRIG _REGULAR_T7_TR GO</code>	TIMER7 TRGO事件（规则组）
<code>ADC0_1_EXTTRIG _REGULAR_EXTI_ 11</code>	外部中断线11（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T2_CH0</code>	TIMER2 CH0事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T1_CH2</code>	TIMER1 CH2事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T0_CH2</code>	TIMER0 CH2事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T7_CH0</code>	TIMER7 CH0事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T7_TRG 0</code>	TIMER7 TRGO事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T4_CH0</code>	TIMER4 CH0事件（规则组）
<code>ADC2_EXTTRIG_R EGULAR_T4_CH2</code>	TIMER4 CH2事件（规则组）
<code>ADC0_1_2_EXTTRI G_REGULAR_NON E</code>	软件触发（规则组）
<code>ADC0_1_EXTTRIG _INSERTED_T0_T RG0</code>	TIMER0 TRGO事件（注入组）
<code>ADC0_1_EXTTRIG _INSERTED_T0_C H3</code>	TIMER0 CH3事件（注入组）
<code>ADC0_1_EXTTRIG</code>	TIMER1 TRGO事件（注入组）

<i>_INSERTED_T1_T</i> RGO	
<i>ADC0_1_EXTTRIG</i> <i>_INSERTED_T1_C</i> H0	TIMER1 CH0事件（注入组）
<i>ADC0_1_EXTTRIG</i> <i>_INSERTED_T2_C</i> H3	TIMER2 CH3事件（注入组）
<i>ADC0_1_EXTTRIG</i> <i>_INSERTED_T3_T</i> RGO	TIMER3 TRGO事件（注入组）
<i>ADC0_1_EXTTRIG</i> <i>_INSERTED_EXTI</i> 15	外部中断线15（注入组）
<i>ADC0_1_EXTTRIG</i> <i>_INSERTED_T7_C</i> H3	TIMER7 CH3事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T0_TRG</i> O	TIMER0 TRGO事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T0_CH3</i>	TIMER0 CH3事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T3_CH2</i>	TIMER3 CH2事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T7_CH1</i>	TIMER7 CH1事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T7_CH3</i>	TIMER7 CH3事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T4_TRG</i> O	TIMER4 TRGO事件（注入组）
<i>ADC2_EXTTRIG_IN</i> <i>SERTED_T4_CH3</i>	TIMER4 CH3事件（注入组）
<i>ADC0_1_2_EXTTRIG_INSERTED_NON</i> E	软件触发（注入组）

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 regular channel external trigger source */

adc_external_trigger_source_config(ADC0,ADC_REGULAR_CHANNEL,
ADC0_1_EXTTRIG_REGULAR_T0_CH0);
```

函数 **adc_software_trigger_enable**

函数adc_software_trigger_enable描述见下表：

表 3-23. 函数 **adc_software_trigger_enable**

函数名称	adc_software_trigger_enable
函数原形	void adc_software_trigger_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	ADC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_channel_group	通道组选择
ADC_REGULAR_CHANNEL	规则通道组
ADC_INSERTED_CHANNEL	注入通道组
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable ADC0 regular channel group software trigger */
adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 **adc_regular_data_read**

函数adc_inserted_regular_data_read描述见下表：

表 3-24. 函数 **adc_regular_data_read**

函数名称	adc_regular_data_read
函数原形	uint16_t adc_regular_data_read(uint32_t adc_periph);
功能描述	读ADC规则组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值 (0-0xFFFF)

例如：

```
/* read ADC0 regular group data register */
uint16_t adc_value = 0;
adc_value = adc_regular_data_read(ADC0);
```

函数 **adc_inserted_data_read**

函数adc_inserted_regular_data_read描述见下表：

表 3-25. 函数 **adc_inserted_data_read**

函数名称	adc_inserted_data_read
函数原形	uint16_t adc_inserted_data_read(uint32_t adc_periph, uint8_t

	inserted_channel);
功能描述	读ADC注入组数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
inserted_channel	注入通道选择
ADC_INSERTED_CHANNEL_x(x=0..3)	注入通道x, x=0,1,2,3
输出参数{out}	
-	-
返回值	
uint16_t	ADC转换值(0-0xFFFF)

例如：

```
/* read ADC0 inserted group data register */
uint16_t adc_value = 0;
adc_value = adc_inserted_data_read (ADC0, ADC_INSERTED_CHANNEL_0);
```

函数 **adc_sync_mode_convert_value_read**

函数adc_sync_mode_convert_value_read描述见下表：

表 3-26. 函数 **adc_sync_mode_convert_value_read**

函数名称	adc_sync_mode_convert_value_read
函数原形	uint32_t adc_sync_mode_convert_value_read(void);
功能描述	在同步模式下，读ADC0和ADC1最近的一次转换结果
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
uint32_t	ADC转换值 (0-0xFFFFFFFF)

例如：

```
/* read the last ADC0 and ADC1 conversion result data in sync mode */  
  
Uint32_t adc_value = 0;  
  
adc_value = adc_sync_mode_convert_value_read();
```

函数 adc_flag_get

函数adc_flag_get描述见下表：

表 3-27. 函数 adc_flag_get

函数名称	adc_flag_get
函数原形	FlagStatus adc_flag_get(uint32_t adc_periph , uint32_t adc_flag);
功能描述	获取ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
ADC_FLAG_EOC	组转换结束标志位
ADC_FLAG_EOIC	注入通道组转换结束标志位
ADC_FLAG_STIC	注入通道组转换开始标志位
ADC_FLAG_STRC	规则通道组转换开始标志位
输出参数{out}	
-	-

返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog flag bits*/
FlagStatus flag_value;
flag_value = adc_flag_get(ADC0, ADC_FLAG_WDE);
```

函数 **adc_flag_clear**

函数adc_flag_clear描述见下表：

表 3-28. 函数 **adc_flag_clear**

函数名称	adc_flag_clear
函数原形	void adc_flag_clear(uint32_t adc_periph, uint32_t adc_flag);
功能描述	清除ADC标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_flag	ADC标志位
ADC_FLAG_WDE	模拟看门狗事件标志位
ADC_FLAG_EOC	组转换结束标志位
ADC_FLAG_EOIC	注入通道组转换结束标志位
ADC_FLAG_STIC	注入通道组转换开始标志位
ADC_FLAG_STRC	规则通道组转换开始标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the ADC0 analog watchdog flag bits*/
```

```
adc_flag_clear(ADC0, ADC_FLAG_WDE);
```

函数 adc_interrupt_flag_get

函数adc_interrupt_flag_get描述见下表：

表 3-29. 函数 adc_interrupt_flag_get

函数名称	adc_interrupt_flag_get
函数原形	FlagStatus adc_interrupt_flag_get(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	获取ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the ADC0 analog watchdog interrupt bits*/
```

```
FlagStatus flag_value;
```

```
flag_value = adc_interrupt_flag_get(ADC0, ADC_INT_WDE);
```

函数 adc_interrupt_flag_clear

函数adc_interrupt_flag_clear描述见下表:

表 3-30. 函数 adc_interrupt_flag_clear

函数名称	adc_interrupt_flag_clear
函数原形	void adc_interrupt_flag_clear(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	清除ADC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_FLAG_WDE	模拟看门狗中断标志位
ADC_INT_FLAG_EOC	组转换结束中断标志位
ADC_INT_FLAG_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear the ADC0 analog watchdog interrupt bits*/
adc_interrupt_flag_clear(ADC0, ADC_INT_WDE);
```

函数 adc_interrupt_enable

函数adc_interrupt_enable描述见下表:

表 3-31. 函数 adc_interrupt_enable

函数名称	adc_interrupt_enable
函数原形	void adc_interrupt_enable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断使能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable ADC0 analog watchdog interrupt */
adc_interrupt_enable(ADC0, ADC_INT_WDE);
```

函数 adc_interrupt_disable

函数adc_interrupt_disable描述见下表：

表 3-32. 函数 adc_interrupt_disable

函数名称	adc_interrupt_disable
函数原形	void adc_interrupt_disable(uint32_t adc_periph, uint32_t adc_interrupt);
功能描述	ADC中断禁能
先决条件	-

被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
adc_interrupt	ADC中断标志位
ADC_INT_WDE	模拟看门狗中断标志位
ADC_INT_EOC	组转换结束中断标志位
ADC_INT_EOIC	注入通道组转换结束中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 interrupt */
adc_interrupt_disable(ADC0, ADC_INT_WDE);
```

函数 **adc_watchdog_single_channel_enable**

函数adc_watchdog_single_channel_enable描述见下表：

表 3-33. 函数 **adc_watchdog_single_channel_enable**

函数名称	adc_watchdog_single_channel_enable
函数原形	void adc_watchdog_single_channel_enable(uint32_t adc_periph, uint8_t adc_channel);
功能描述	配置ADC模拟看门狗单通道有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择

输入参数{in}	
adc_channel	选择ADC通道
<i>ADC_CHANNEL_x(x=0..17)</i>	ADC Channelx(x=0..17) (只有ADC0, 可取值x=16和17)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 analog watchdog single channel */
adc_watchdog_single_channel_enable(ADC0, ADC_CHANNEL_1);
```

函数 **adc_watchdog_group_channel_enable**

函数adc_watchdog_group_channel_enable描述见下表：

表 3-34. 函数 **adc_watchdog_group_channel_enable**

函数名称	adc_watchdog_group_channel_enable
函数原形	void adc_watchdog_group_channel_enable(uint32_t adc_periph, uint8_t adc_channel_group);
功能描述	配置ADC模拟看门狗在通道组有效
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输入参数{in}	
adc_channel_group	通道组使用模拟看门狗
<i>ADC_REGULAR_CHANNEL</i>	规则通道组
<i>ADC_INSERTED_CHANNEL</i>	注入通道组

<i>ADC_REGULAR_IN SERTED_CHANNEL_L</i>	规则和注入通道组
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ADC0 analog watchdog group channel */
adc_watchdog_group_channel_enable(ADC0, ADC_REGULAR_CHANNEL);
```

函数 **adc_watchdog_disable**

函数adc_watchdog_disable描述见下表：

表 3-35. 函数 **adc_watchdog_disable**

函数名称	adc_watchdog_disable
函数原形	void adc_watchdog_disable(uint32_t adc_periph);
功能描述	ADC模拟看门狗禁能
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
<i>ADCx(x=0..2)</i>	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable ADC0 analog watchdog */
adc_watchdog_disable(ADC0);
```

函数 adc_watchdog_threshold_config

函数adc_watchdog_threshold_config描述见下表:

表 3-36. 函数 adc_watchdog_threshold_config

函数名称	adc_watchdog_threshold_config
函数原形	void adc_watchdog_threshold_config(uint32_t adc_periph, uint16_t low_threshold, uint16_t high_threshold);
功能描述	配置ADC模拟看门狗阈值
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输入参数{in}	
low_threshold	模拟看门狗低阈值, 0..4095
输入参数{in}	
high_threshold	模拟看门狗高阈值, 0..4095
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure ADC0 analog watchdog threshold */
adc_watchdog_threshold_config(ADC0, 0x0400, 0x0A00);
```

函数 adc_oversample_mode_config

函数adc_oversample_mode_config描述见下表:

表 3-37. 函数 adc_oversample_mode_config

函数名称	adc_oversample_mode_config
函数原形	void adc_oversample_mode_config(uint32_t adc_periph, uint32_t mode, uint16_t

	shift, uint8_t ratio);
功能描述	配置ADCx过采样模式
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=..2)	ADC外设选择
输入参数{in}	
mode	ADC过采样触发模式
ADC_OVERSA MPLING_ALL_C ONVERT	在一个触发之后，对一个通道连续进行过采样转换
ADC_OVERSA MPLING_ONE_ CONVERT	在一个触发之后，对一个通道只进行一次过采样转换
输入参数{in}	
shift	ADC过滤采样移位
ADC_OVERSA MPLING_SHIFT _NONE	不移位
ADC_OVERSA MPLING_SHIFT _1B	移1位
ADC_OVERSA MPLING_SHIFT _2B	移2位
ADC_OVERSA MPLING_SHIFT _3B	移3位
ADC_OVERSA MPLING_SHIFT _4B	移4位
ADC_OVERSA MPLING_SHIFT	移5位

<i>_5B</i>	
<i>ADC_OVERSA</i> <i>MPLING_SHIFT</i> <i>_6B</i>	移6位
<i>ADC_OVERSA</i> <i>MPLING_SHIFT</i> <i>_7B</i>	移7位
<i>ADC_OVERSA</i> <i>MPLING_SHIFT</i> <i>_8B</i>	移8位
输入参数{in}	
ratio	ADC过采样率
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL2</i>	2x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL4</i>	4x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL8</i>	8x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL16</i>	16x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL32</i>	32x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL64</i>	64x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL128</i>	128x
<i>ADC_OVERSA</i> <i>MPLING_RATIO</i> <i>_MUL256</i>	256x

输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* configure ADC1 oversample mode: 16 times sample, 4 bits shift */

adc_oversample_mode_config(ADC1, ADC_OVERSAMPLING_ALL_CONVERT,
    ADC_OVERSAMPLING_SHIFT_4B, ADC_OVERSAMPLING_RATIO_MUL16);
```

函数 **adc_oversample_mode_enable**

函数adc_oversample_mode_enable描述见下表:

表 3-38. 函数 **adc_oversample_mode_enable**

函数名称	adc_oversample_mode_enable
函数原形	void adc_oversample_mode_enable(uint32_t adc_periph);
功能描述	使能ADCx过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* enable ADC0 oversample mode */

adc_oversample_mode_enable (ADC0);
```

函数 **adc_oversample_mode_disable**

函数adc_oversample_mode_disable描述见下表:

表 3-39. 函数 adc_oversample_mode_disable

函数名称	adc_oversample_mode_disable
函数原形	void adc_oversample_mode_disable(uint32_t adc_periph);
功能描述	禁能ADCx过采样
先决条件	-
被调用函数	-
输入参数{in}	
adc_periph	ADC外设
ADCx(x=0..2)	ADC外设选择
输出参数{out}	
-	-
返回值	
-	-

Example:

```
/* disable ADC0 oversample mode */
adc_oversample_mode_disable (ADC0);
```

3.3. BKP

位于备份域中的备份寄存器可在V_{DD}电源关闭时由V_{BAT}供电，备份寄存器有42个16位(84字节)寄存器可用来存储并保护用户应用数据，从待机模式唤醒或系统复位也不会对这些寄存器造成影响。章节[3.3.1](#)描述了BKP的寄存器列表，章节[3.3.2](#)对BKP库函数进行说明。

3.3.1. 外设寄存器说明

BKP寄存器列表如下表所示：

表 3-40. BKP 寄存器

寄存器名称	寄存器描述
BKP_DATAx (x=0..41)	备份数据寄存器
BKP_OCTL	RTC信号输出控制寄存器
BKP_TPCTL	侵入引脚控制寄存器

寄存器名称	寄存器描述
BKP_TPCS	侵入控制状态寄存器

3.3.2. 外设库函数说明

BKP库函数列表如下表所示：

表 3-41. BKP 库函数

库函数名称	库函数描述
bkp_deinit	复位备份数据寄存器
bkp_write_data	写备份数据寄存器
bkp_read_data	读备份数据寄存器
bkp_rtc_calibration_output_enable	RTC时钟校准输出使能
bkp_rtc_calibration_output_disable	RTC时钟校准输出失能
bkp_rtc_signal_output_enable	RTC闹钟或秒信号输出使能
bkp_rtc_signal_output_disable	RTC闹钟或秒信号输出失能
bkp_rtc_output_select	RTC输出选择， RTC输出可选择为闹钟脉冲或秒脉冲
bkp_rtc_clock_output_select	RTC时钟输出选择
bkp_rtc_clock_calibration_direction	RTC时钟校准方向选择
bkp_rtc_calibration_value_set	RTC时钟校准值
bkp_tamper_detection_enable	TAMPER引脚使能
bkp_tamper_detection_disable	TAMPER引脚失能
bkp_tamper_active_level_set	TAMPER引脚有效电平设置
bkp_tamper_interrupt_enable	TAMPER中断使能
bkp_tamper_interrupt_disable	TAMPER中断失能
bkp_flag_get	获取标志位
bkp_flag_clear	清除标志位
bkp_interrupt_flag_get	获取中断标志位
bkp_interrupt_flag_clear	清除中断标志位

函数 bkp_deinit

函数 bkp_deinit 描述见下表:

表 3-42. 函数 bkp_deinit

函数名称	bkp_deinit
函数原型	void bkp_deinit(void);
功能描述	复位备份数据寄存器
先决条件	-
被调用函数	rcu_bkp_reset_enable / rcu_bkp_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset BKP registers */

bkp_deinit();
```

函数 bkp_write_data

函数 bkp_write_data 描述见下表:

表 3-43. 函数 bkp_write_data

函数名称	bkp_write_data
函数原型	void bkp_write_data(bkp_data_register_enum register_number, uint16_t data);
功能描述	写备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举 bkp_data_register_enum
BKP_DATA_x(x =	BKP 数据寄存器 x

0..41)	
输入参数{in}	
Data	待写入BKP数据寄存器的数据
0-0xffff	数值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write BKP data register */

bkp_write_data (BKP_DATA_0, 0x1226);
```

函数 bkp_read_data

函数bkp_read_data描述见下表：

表 3-44. 函数 bkp_read_data

函数名称	bkp_read_data
函数原型	uint16_t bkp_read_data(bkp_data_register_enum register_number);
功能描述	读备份数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
register_number	参考枚举bkp_data_register_enum
<i>BKP_DATA_x(x = 0..41)</i>	BKP数据寄存器x
输出参数{out}	
-	-
返回值	
uint16_t	0-0xffff

例如：

```
/* read BKP data register */
```

```

uint16_t data;
data = bkp_read_data (BKP_DATA_0);

```

函数 bkp_rtc_calibration_output_enable

函数bkp_rtc_calibration_output_enable描述见下表:

表 3-45. 函数 bkp_rtc_calibration_output_enable

函数名称	bkp_rtc_calibration_output_enable
函数原型	void bkp_rtc_calibration_output_enable(void);
功能描述	RTC时钟校准输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable RTC clock calibration output */
bkp_rtc_calibration_output_enable();

```

函数 bkp_rtc_calibration_output_disable

函数bkp_rtc_calibration_output_disable描述见下表:

表 3-46. 函数 bkp_rtc_calibration_output_disable

函数名称	bkp_rtc_calibration_output_disable
函数原型	void bkp_rtc_calibration_output_disable(void);
功能描述	RTC时钟校准输出失能
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC clock calibration output */

bkp_rtc_calibration_output_disable();
```

函数 bkp_rtc_signal_output_enable

函数bkp_rtc_signal_output_enable描述见下表：

表 3-47. 函数 bkp_rtc_signal_output_enable

函数名称	bkp_rtc_signal_output_enable
函数原型	void bkp_rtc_signal_output_enable (void);
功能描述	RTC闹钟或秒信号输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RTC alarm or second signal output */

bkp_rtc_signal_output_enable();
```

函数 bkp_rtc_signal_output_disable

函数bkp_rtc_signal_output_disable描述见下表：

表 3-48. 函数 bkp_rtc_signal_output_disable

函数名称	bkp_rtc_signal_output_disable
函数原型	void bkp_rtc_signal_output_disable (void);
功能描述	RTC闹钟或秒信号输出失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable RTC alarm or second signal output */
bkp_rtc_signal_output_disable();
```

函数 bkp_rtc_output_select

函数 bkp_rtc_output_select 描述见下表：

表 3-49. 函数 bkp_rtc_output_select

函数名称	bkp_rtc_output_select
函数原型	void bkp_rtc_output_select (uint16_t outputsel);
功能描述	RTC输出选择， RTC输出可选择为闹钟脉冲或秒脉冲
先决条件	-
被调用函数	-
输入参数{in}	
outputsel	RTC输出选择
<i>RTC_OUTPUT_ALARM_PULSE</i>	RTC闹钟脉冲被选择为RTC输出
<i>RTC_OUTPUT_SECOND_PULSE</i>	RTC秒脉冲被选择为RTC输出

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC output alarm signal output */
bkp_rtc_output_select(RTC_OUTPUT_ALARM_PULSE);
```

函数 bkp_rtc_clock_output_select

函数bkp_rtc_clock_output_select描述见下表：

表 3-50. 函数 bkp_rtc_clock_output_select

函数名称	bkp_rtc_clock_output_select
函数原型	void bkp_rtc_clock_output_select(uint16_t clocksel);
功能描述	RTC时钟输出选择， RTC时钟输出可选择为不分频或64分频
先决条件	-
被调用函数	-
输入参数{in}	
clocksel	RTC时钟输出选择
RTC_CLOCK_DIV_64	RTC时钟输出被选择为64分频
RTC_CLOCK_DIV_1	RTC时钟输出被选择为不分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select RTC clock devided 64 to output */
bkp_rtc_clock_output_select(RTC_CLOCK_DIV_64);
```

函数 bkp_rtc_clock_calibration_direction

函数 bkp_rtc_clock_calibration_direction 描述见下表：

表 3-51. 函数 bkp_rtc_clock_calibration_direction

函数名称	bkp_rtc_clock_calibration_direction
函数原型	void bkp_rtc_clock_calibration_direction (uint16_t direction);
功能描述	RTC时钟校准方向选择， RTC时钟校准方向可选择为变快或变慢
先决条件	-
被调用函数	-
输入参数{in}	
direction	RTC时钟校准方向
RTC_CLOCK_SLOWED_DOWN	RTC时钟变慢
RTC_CLOCK_SPEED_UP	RTC时钟变快
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC clock slowed down */
bkp_rtc_clock_calibration_direction (RTC_CLOCK_SLOWED_DOWN);
```

函数 bkp_rtc_calibration_value_set

函数 bkp_rtc_calibration_value_set 描述见下表：

表 3-52. 函数 bkp_rtc_calibration_value_set

函数名称	bkp_rtc_calibration_value_set
函数原型	void bkp_rtc_calibration_value_set(uint8_t value);
功能描述	RTC时钟校准值
先决条件	-
被调用函数	-

输入参数{in}	
value	RTC时钟校准值
0x00 - 0x7F	校准值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RTC clock calibration value */
bkp_rtc_calibration_value_set (0x7f);
```

函数 **bkp_tamper_detection_enable**

函数**bkp_tamper_detection_enable**描述见下表：

表 3-53. 函数 **bkp_tamper_detection_enable**

函数名称	bkp_tamper_detection_enable
函数原型	void bkp_tamper_detection_enable (void);
功能描述	TAMPER引脚使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable tamper pin detection */
bkp_tamper_detection_enable();
```

函数 bkp_tamper_detection_disable

函数 bkp_tamper_detection_disable 描述见下表：

表 3-54. 函数 bkp_tamper_detection_disable

函数名称	bkp_tamper_detection_disable
函数原型	void bkp_tamper_detection_disable (void);
功能描述	TAMPER引脚失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable tamper pin detection */
bkp_tamper_detection_disable();
```

函数 bkp_tamper_active_level_set

函数 bkp_tamper_active_level_set 描述见下表：

表 3-55. 函数 bkp_tamper_active_level_set

函数名称	bkp_tamper_active_level_set
函数原型	void bkp_tamper_active_level_set (uint16_t level);
功能描述	TAMPER引脚有效电平设置
先决条件	-
被调用函数	-
输入参数{in}	
level	TAMPER引脚有效电平
TAMPER_PIN_ACT	TAMPER引脚高电平有效

<i>IVE_HIGH</i>	
<i>TAMPER_PIN_ACT</i>	
<i>IVE_LOW</i>	TAMPER引脚低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set tamper pin active level high */
bkp_tamper_active_level_set (TAMPER_PIN_ACTIVE_HIGH);
```

函数 bkp_tamper_interrupt_enable

函数bkp_tamper_interrupt_enable描述见下表：

表 3-56. 函数 bkp_tamper_interrupt_enable

函数名称	bkp_tamper_interrupt_enable
函数原型	void bkp_tamper_interrupt_enable (void);
功能描述	TAMPER中断使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable tamper pin interrupt */
bkp_tamper_interrupt_enable ();
```

函数 bkp_tamper_interrupt_disable

函数 bkp_tamper_interrupt_disable 描述见下表：

表 3-57. 函数 bkp_tamper_interrupt_disable

函数名称	bkp_tamper_interrupt_disable
函数原型	void bkp_tamper_interrupt_disable (void);
功能描述	TAMPER 中断失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable tamper pin interrupt */
bkp_tamper_interrupt_disable();
```

函数 bkp_flag_get

函数 bkp_flag_get 描述见下表：

表 3-58. 函数 bkp_flag_get

函数名称	bkp_flag_get
函数原型	FlagStatus bkp_flag_get(uint16_t flag);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
BKP_FLAG_TAMP ER	侵入事件标志

输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get BKP flag state */

FlagStatus status;

status = bkp_flag_get (BKP_FLAG_TAMPER);
```

函数 bkp_flag_clear

函数 bkp_flag_clear 描述见下表：

表 3-59. 函数 bkp_flag_clear

函数名称	bkp_flag_clear
函数原型	void bkp_flag_clear(uint16_t flag);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
BKP_FLAG_TAMP ER	侵入事件标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear BKP flag state */

bkp_flag_clear (BKP_FLAG_TAMPER);
```

函数 bkp_interrupt_flag_get

函数 bkp_interrupt_flag_get 描述见下表：

表 3-60. 函数 bkp_interrupt_flag_get

函数名称	bkp_interrupt_flag_get
函数原型	FlagStatus bkp_interrupt_flag_get(uint16_t flag);
功能描述	获取中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
BKP_INT_FLAG_T AMPER	侵入事件中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get BKP interrupt flag state */

bkp_interrupt_flag_get (BKP_FLAG_TAMPER);
```

函数 bkp_interrupt_flag_clear

函数 bkp_interrupt_flag_clear 描述见下表：

表 3-61. 函数 bkp_interrupt_flag_clear

函数名称	bkp_interrupt_flag_clear
函数原型	void bkp_interrupt_flag_clear(uint16_t flag);
功能描述	清除中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
BKP_INT_FLAG_T AMPER	侵入事件中断标志
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* clear BKP interrupt flag state */  
bkp_interrupt_flag_clear (BKP_INT_FLAG_TAMPER);
```

3.4. CAN

CAN (Controller Area Network) 总线是一种可以在无主机情况下实现微处理器或者设备之间相互通信的总线标准。章节[3.4.1](#)描述了CAN的寄存器列表，章节[3.4.2](#)对CAN库函数进行说明

3.4.1. 外设寄存器说明

CAN寄存器列表如下表所示：

表 3-62. CAN 寄存器

寄存器名称	寄存器描述
CAN_CTL	控制寄存器
CAN_STAT	状态寄存器
CAN_TSTAT	发送状态寄存器
CAN_RFIFO0	接收FIFO0寄存器
CAN_RFIFO1	接收FIFO1寄存器
CAN_INTEN	中断使能寄存器
CAN_ERR	错误寄存器
CAN_BT	位时序寄存器
CAN_TMIx	发送邮箱标识符寄存器
CAN_TMPx	发送邮箱属性寄存器
CAN_TMDATA0x	发送邮箱data0寄存器
CAN_TMDATA1x	发送邮箱data1寄存器
CAN_RFIFOMIx	接收FIFO邮箱标识符寄存器
CAN_RFIFOMPx	接收FIFO邮箱属性寄存器
CAN_RFIFOMDAT A0x	接收FIFO邮箱data0寄存器
CAN_RFIFOMDAT A1x	接收FIFO邮箱data1寄存器
CAN_FCTL	过滤器控制寄存器
CAN_FMCFG	过滤器模式配置寄存器
CAN_FSCFG	过滤器位宽配置寄存器

寄存器名称	寄存器描述
CAN_FAFIFO	过滤器关联FIFO寄存器
CAN_FW	过滤器激活寄存器
CAN_FxDATAy	过滤器(x)数据(y)寄存器

3.4.2. 外设库函数说明

CAN库函数列表如下表所示：

表 3-63. CAN 库函数

库函数名称	库函数描述
can_deinit	复位外设CAN
can_struct_para_init	初始化结构体
can_init	初始化外设CAN
can_filter_init	CAN过滤器初始化
can1_filter_start_bank	CAN1过滤器序起始编号设置
can_debug_freeze_enable	CAN调试冻结使能
can_debug_freeze_disable	CAN调试冻结关闭
can_time_trigger_mode_enable	CAN时间触发模式使能
can_time_trigger_mode_disable	CAN时间触发模式关闭
can_message_transmit	CAN传输报文
can_transmit_states	获取CAN传输状态
can_transmission_stop	CAN邮箱停止发送
can_message_receive	CAN接收报文
can_fifo_release	CAN释放FIFO
can_receive_message_length_get	获取CAN接收帧的数量
can_working_mode_set	CAN工作模式设置
can_wakeup	从睡眠模式中唤醒CAN
can_error_get	获取CAN总线错误
can_receive_error_number_get	获取CAN接收错误
can_transmit_error_number_get	获取CAN发送错误

库函数名称	库函数描述
can_interrupt_enable	CAN中断使能
can_interrupt_disable	CAN中断关闭
can_flag_get	获取CAN标志位状态
can_flag_clear	清除CAN标志位状态
can_interrupt_flag_get	获取CAN中断标志位状态
can_interrupt_flag_clear	清除CAN中断标志位状态

结构体 can_parameter_struct

表 3-64. 结构体 can_parameter_struct

成员名称	功能描述
working_mode	工作模式
resync_jump_width	再同步补偿宽度
time_segment_1	位段1
time_segment_2	位段2
time_triggered	时间触发通信模式
auto_bus_off_recovery	自动离线恢复
auto_wake_up	自动唤醒
auto_retrans	自动重传
rec_fifo_overwrite	接收FIFO满时覆盖
trans_fifo_order	发送FIFO顺序
prescaler	波特率分频系数

结构体 can_transmit_message_struct

表 3-65. 结构体 can_transmit_message_struct

成员名称	功能描述
tx_sfid	标准格式帧标识符
tx_efid	扩展格式帧标识符
tx_ff	帧格式： 标准格式/扩展格式

tx_ft	帧类型: 数据帧/远程帧
tx_dlen	数据长度
tx_data[8]	数据值

结构体 can_receive_message_struct

表 3-66. 结构体 can_receive_message_struct

成员名称	功能描述
rx_sfid	标准格式帧标识符
rx_efid	扩展格式帧标识符
rx_ff	帧格式: 标准格式/扩展格式
rx_ft	帧类型: 数据帧/远程帧
rx_dlen	数据长度
rx_data[8]	数据值
rx_fi	过滤器索引

结构体 can_filter_parameter_struct

表 3-67. 结构体 can_filter_parameter_struct

成员名称	功能描述
filter_list_high	过滤器列表数高位
filter_list_low	过滤器列表数低位
filter_mask_high	过滤器掩码数高位
filter_mask_low	过滤器掩码数低位
filter_fifo_number	接收FIFO编号
filter_number	过滤器索引号
filter_mode	过滤模式: 列表模式/掩码模式
filter_bits	过滤器位宽
filter_enable	过滤器是否工作

函数 can_deinit

函数can_deinit描述见下表:

表 3-68. 函数 can_deinit

函数名称	can_deinit
函数原型	void can_deinit(uint32_t can_periph);
功能描述	复位外设CAN
先决条件	-
被调用函数	rcu_periph_reset_enable/ rcu_periph_reset_disable
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 deinitialize*/
can_deinit (CAN0);
```

函数 can_struct_para_init

函数can_struct_para_init描述见下表：

表 3-69. 函数 can_struct_para_init

函数名称	can_struct_para_init
函数原型	void can_struct_para_init(can_struct_type_enum type, void* p_struct)
功能描述	CAN外设库使用到的各类结构体初始化
先决条件	-
被调用函数	-
输入参数{in}	
type	需要初始化的结构体类型, 仅可选择唯一参数
CAN_INIT_STRUCT_T	初始化结构体
CAN_FILTER_STR	过滤器初始化结构体

<i>UCT</i>	
<i>CAN_TX_MESSAGE_STRUCT</i>	存储发送帧结构体
<i>CAN_RX_MESSAGE_STRUCT</i>	接收帧结构体
输出参数{out}	
<i>p_struct</i>	对应的需要初始化的结构体指针
返回值	
-	-

例如：

```
can_parameter_struct can_init;
can_struct_para_init (CAN_INIT_STRUCT, &can_init);
```

函数 **can_init**

函数can_init描述见下表：

表 3-70. 函数 **can_init**

函数名称	can_init
函数原型	ErrStatus can_init(uint32_t can_periph, can_parameter_struct* can_parameter_init);
功能描述	初始化外设CAN
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择， CAN1仅适用于GD32F30X_CL
输入参数{in}	
can_parameter_init	初始化结构体，结构体成员参考 表 3-64. 结构体can_parameter_struct
输出参数{out}	
-	-
返回值	

ErrStatus	SUCCESS / ERROR
-----------	-----------------

例如：

```
/* CAN0 initialize*/
can_init(CAN0);
```

函数 can_filter_init

函数can_filter_init描述见下表：

表 3-71. 函数 can_filter_init

函数名称	can_filter_init
函数原型	void can_filter_init(can_filter_parameter_struct* can_filter_parameter_init);
功能描述	CAN过滤器初始化
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_filter_parameter_init	过滤器初始化结构体，结构体成员参考 表 3-67. 结构体 can_filter_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize CAN filter */
can_filter_init(&can_filter);
```

函数 can1_filter_start_bank

函数can1_filter_start_bank描述见下表：

表 3-72. 函数 can1_filter_start_bank

函数名称	can1_filter_start_bank
函数原型	void can1_filter_start_bank(uint8_t start_bank);
功能描述	CAN1过滤器序起始编号设置

先决条件	-
被调用函数	-
输入参数{in}	
start_bank	CAN1过滤器序起始编号
1..27	可选的编号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set CAN1 fliter start bank number 15*/
can1_filter_start_bank (15);
```

函数 can_debug_freeze_enable

函数can_debug_freeze_enable描述见下表：

表 3-73. 函数 can_debug_freeze_enable

函数名称	can_debug_freeze_enable
函数原型	void can_debug_freeze_enable(uint32_t can_periph);
功能描述	CAN调试冻结使能
先决条件	-
被调用函数	dbg_periph_enable
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择， CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enable CAN0 debug freeze */

can_debug_freeze_enable (CAN0);

```

函数 can_debug_freeze_disable

函数can_debug_freeze_disable描述见下表:

表 3-74. 函数 can_debug_freeze_disable

函数名称	can_debug_freeze_disable
函数原型	void can_debug_freeze_disable(uint32_t can_periph);
功能描述	CAN调试冻结关闭
先决条件	-
被调用函数	dbg_periph_disable
输入参数{in}	
can_periph	CAN 外设
CANx(x=0, 1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* disable CAN0 debug freeze */

can_debug_freeze_disable (CAN0);

```

函数 can_time_trigger_mode_enable

函数can_time_trigger_mode_enable描述见下表:

表 3-75. 函数 can_time_trigger_mode_enable

函数名称	can_time_trigger_mode_enable
函数原型	void can_time_trigger_mode_enable(uint32_t can_periph);
功能描述	CAN时间触发模式使能
先决条件	-
被调用函数	-

输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CAN0 time trigger mode */
can_time_trigger_mode_enable (CAN0);
```

函数 can_time_trigger_mode_disable

函数can_time_trigger_mode_disable描述见下表：

表 3-76. 函数 can_time_trigger_mode_disable

函数名称	can_time_trigger_mode_disable
函数原型	void can_time_trigger_mode_disable(uint32_t can_periph);
功能描述	CAN时间触发模式关闭
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CAN0 time trigger mode */
can_time_trigger_mode_disable (CAN0);
```

函数 can_message_transmit

函数can_message_transmit描述见下表:

表 3-77. 函数 can_message_transmit

函数名称	can_message_transmit
函数原型	uint8_t can_message_transmit(uint32_t can_periph, can_trasnmit_message_struct* transmit_message);
功能描述	CAN传输报文
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
transmit_message	报文发送结构体, 结构体成员参考 表 3-65. 结构体 can_trasnmit_message_struct
输出参数{out}	
-	-
返回值	
uint8_t	0x00-0x03

例如:

```
/* CAN0 transmit message and return the mailbox number */

uint8_t transmit_mailbox = 0;

transmit_mailbox = can_message_transmit(CAN0, &transmit_message);
```

函数 can_transmit_states

函数can_transmit_states描述见下表:

表 3-78. 函数 can_transmit_states

函数名称	can_transmit_states
函数原型	can_transmit_state_enum can_transmit_states(uint32_t can_periph, uint8_t mailbox_number);

功能描述	获取CAN传输状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
can_transmit_state_enum	0..4

例如:

```
/* CAN0 mailbox0 transmit state */

uint8_t transmit_state = 0;

transmit_state = can_transmit_states (CAN0, CAN_MAILBOX0);
```

函数 can_transmission_stop

函数can_transmission_stop描述见下表:

表 3-79. 函数 can_transmission_stop

函数名称	can_transmission_stop
函数原型	void can_transmission_stop(uint32_t can_periph, uint8_t mailbox_number);
功能描述	CAN邮箱停止发送
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设

CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
mailbox_number	邮箱标号
CAN_MAILBOXx	CAN_MAILBOXx(x=0,1,2)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* stop CAN0 mailbox0 transmission */
can_transmission_stop (CAN0, CAN_MAILBOX0);
```

函数 can_message_receive

函数can_message_receive描述见下表:

表 3-80. 函数 can_message_receive

函数名称	can_message_receive
函数原型	void can_message_receive(uint32_t can_periph, uint8_t fifo_number, can_receive_message_struct* receive_message);
功能描述	CAN接收报文
先决条件	can_struct_para_init()
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输入参数{in}	
receive_message	接收报文结构体, 结构体成员参考 表 3-66. 结构体 can_receive_message_struct

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 FIFO0 receive message */
can_message_receive(CAN0, CAN_FIFO0, &receive_message);
```

函数 can_fifo_release

函数can_fifo_release描述见下表：

表 3-81. 函数 can_fifo_release

函数名称	can_fifo_release
函数原型	void can_fifo_release(uint32_t can_periph, uint8_t fifo_number);
功能描述	CAN释放FIFO
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 release FIFO0 */
can_fifo_release (CAN0, CAN_FIFO0);
```

函数 can_receive_message_length_get

函数can_receive_message_length_get描述见下表:

表 3-82. 函数 can_receive_message_length_get

函数名称	can_receive_message_length_get
函数原型	uint8_t can_receive_message_length_get(uint32_t can_periph, uint8_t fifo_number);
功能描述	获取CAN接收帧的数量
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
fifo_number	FIFO编号
CAN_FIFOx	CAN_FIFOx(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	0..3

例如:

```
/* CAN0 FIFO0 receive message length */

uint8_t frame_number = 0;

frame_number = can_receive_message_length_get (CAN0, CAN_FIFO0);
```

函数 can_working_mode_set

函数can_working_mode_set描述见下表:

表 3-83. 函数 can_working_mode_set

函数名称	can_working_mode_set
函数原型	ErrStatus can_working_mode_set(uint32_t can_periph, uint8_t working_mode);
功能描述	CAN工作模式设置

先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
can_working_mod_e	模式选择
CAN_MODE_INITIALIZE	初始化模式
CAN_MODE_NORMAL	正常模式
CAN_MODE_SLEEP	睡眠模式
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如：

```
/* set CAN0 working at initialize mode */
can_working_mode_set (CAN0, CAN_MODE_INITIALIZE);
```

函数 can_wakeup

函数can_wakeup描述见下表：

表 3-84. 函数 can_wakeup

函数名称	can_wakeup
函数原型	ErrStatus can_wakeup(uint32_t can_periph);
功能描述	从睡眠模式中唤醒CAN
先决条件	-
被调用函数	-

输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS / ERROR

例如:

```
/* wake up CAN0 */
can_wakeup(CAN0);
```

函数 can_error_get

函数can_error_get描述见下表:

表 3-85. 函数 can_error_get

函数名称	can_error_get
函数原型	can_error_enum can_error_get(uint32_t can_periph);
功能描述	获取CAN总线错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
can_error_enum	0..7

例如:

```
/* get CAN0 error type */
can_error_enum err_type;
```

```
err_type = can_error_get (CAN0);
```

函数 **can_receive_error_number_get**

函数can_receive_error_number_get描述见下表:

表 3-86. 函数 can_receive_error_number_get

函数名称	can_receive_error_number_get
函数原型	uint8_t can_receive_error_number_get(uint32_t can_periph);
功能描述	获取CAN接收错误
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如:

```
/* get CAN0 receive error number */

uint8_t error_num;

error_num = can_receive_error_number_get (CAN0);
```

函数 **can_transmit_error_number_get**

函数can_transmit_error_number_get描述见下表:

表 3-87. 函数 can_transmit_error_number_get

函数名称	can_transmit_error_number_get
函数原型	uint8_t can_transmit_error_number_get(uint32_t can_periph);
功能描述	获取CAN发送错误
先决条件	-
被调用函数	-

输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输出参数{out}	
-	-
返回值	
uint8_t	0..255

例如：

```
/* get CAN0 transmit error number */

uint8_t error_num;

error_num = can_transmit_error_number_get (CAN0);
```

函数 can_interrupt_enable

函数can_interrupt_enable描述见下表：

表 3-88. 函数 can_interrupt_enable

函数名称	can_interrupt_enable
函数原型	void can_interrupt_enable(uint32_t can_periph, uint32_t interrupt);
功能描述	CAN中断使能
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RF00	接收FIFO0溢出中断使能

<code>CAN_INT_RFNE1</code>	接收FIFO1非空中断使能
<code>CAN_INT_RFF1</code>	接收FIFO1满中断使能
<code>CAN_INT_RFO1</code>	接收FIFO1溢出中断使能
<code>CAN_INT_WERR</code>	警告错误中断使能
<code>CAN_INT_PERR</code>	被动错误中断使能
<code>CAN_INT_BO</code>	离线中断使能
<code>CAN_INT_ERRN</code>	错误种类中断使能
<code>CAN_INT_ERR</code>	错误中断使能
<code>CAN_INT_WU</code>	唤醒中断使能
<code>CAN_INT_SLPW</code>	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 transmit mailbox empty interrupt enable */
can_interrupt_enable (CAN0, CAN_INT_TME);
```

函数 `can_interrupt_disable`

函数`can_interrupt_disable`描述见下表：

表 3-89. 函数 `can_interrupt_disable`

函数名称	<code>can_interrupt_disable</code>
函数原型	<code>void can_interrupt_disable(uint32_t can_periph, uint32_t interrupt);</code>
功能描述	CAN中断关闭
先决条件	-
被调用函数	-
输入参数{in}	
<code>can_periph</code>	CAN 外设
<code>CANx(x=0,1)</code>	CAN外设选择，CAN1仅适用于GD32F30X_CL

输入参数{in}	
interrupt	中断类型
CAN_INT_TME	发送邮箱空中断使能
CAN_INT_RFNE0	接收FIFO0非空中断使能
CAN_INT_RFF0	接收FIFO0满中断使能
CAN_INT_RF00	接收FIFO0溢出中断使能
CAN_INT_RFNE1	接收FIFO1非空中断使能
CAN_INT_RFF1	接收FIFO1满中断使能
CAN_INT_RF01	接收FIFO1溢出中断使能
CAN_INT_WERR	警告错误中断使能
CAN_INT_PERR	被动错误中断使能
CAN_INT_BO	离线中断使能
CAN_INT_ERRN	错误种类中断使能
CAN_INT_ERR	错误中断使能
CAN_INT_WU	唤醒中断使能
CAN_INT_SLPW	睡眠中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CAN0 transmit mailbox empty interrupt disable */
can_interrupt_disable (CAN0, CAN_INT_TME);
```

函数 can_flag_get

函数can_flag_get描述见下表：

表 3-90. 函数 can_flag_get

函数名称	can_flag_get
函数原型	FlagStatus can_flag_get(uint32_t can_periph, can_flag_enum flag);

功能描述	获取CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
flag	CAN 标志位
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO0溢出
CAN_FLAG_RFF0	接收FIFO0满
CAN_FLAG_RFO1	接收FIFO1溢出
CAN_FLAG_RFF1	接收FIFO1满
CAN_FLAG_BOERR	离线错误
CAN_FLAG_PERR	被动错误
CAN_FLAG_WERR	警告错误
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如:

```
/* get CAN0 mailbox 0 transmit finished flag */

can_flag_get (CAN0, CAN_FLAG_MTF0);
```

函数 can_flag_clear

函数can_flag_clear描述见下表:

表 3-91. 函数 can_flag_clear

函数名称	can_flag_clear
函数原型	void can_flag_clear(uint32_t can_periph, can_flag_enum flag);
功能描述	清除CAN标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
flag	CAN 标志位
CAN_FLAG_MTE2	邮箱2发送错误
CAN_FLAG_MTE1	邮箱1发送错误
CAN_FLAG_MTE0	邮箱0发送错误
CAN_FLAG_MTF2	邮箱2发送完成
CAN_FLAG_MTF1	邮箱1发送完成
CAN_FLAG_MTF0	邮箱0发送完成
CAN_FLAG_RFO0	接收FIFO溢出
CAN_FLAG_RFF0	接收FIFO满
CAN_FLAG_RFO1	接收FIFO溢出
CAN_FLAG_RFF1	接收FIFO满
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear CAN0 mailbox 0 transmit error flag */
```

```
can_flag_clear (CAN0, CAN_FLAG_MTE0);
```

函数 can_interrupt_flag_get

函数can_interrupt_flag_get描述见下表:

表 3-92. 函数 can_interrupt_flag_get

函数名称	can_interrupt_flag_get
函数原型	FlagStatus can_interrupt_flag_get(uint32_t can_periph, can_interrupt_flag_enum flag);
功能描述	获取CAN中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
CANx(x=0,1)	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
flag	CAN中断标志位
CAN_INT_FLAG_S LPIF	进入睡眠工作模式的状态改变中断标志
CAN_INT_FLAG_W UIF	从睡眠工作模式唤醒的状态改变中断标志
CAN_INT_FLAG_E RRIF	错误中断标志
CAN_INT_FLAG_M TF2	邮箱2发送完成中断标志
CAN_INT_FLAG_M TF1	邮箱1发送完成中断标志
CAN_INT_FLAG_M TF0	邮箱0发送完成中断标志
CAN_INT_FLAG_R FO0	接收FIFO溢出中断标志
CAN_INT_FLAG_R FF0	接收FIFO满中断标志

<i>CAN_INT_FLAG_R</i>	接收FIFO1溢出中断标志
<i>CAN_INT_FLAG_R</i>	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* get CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_get (CAN0, CAN_INT_FLAG_MTF0);
```

函数 can_interrupt_flag_clear

函数can_interrupt_flag_clear描述见下表：

表 3-93. 函数 can_interrupt_flag_clear

函数名称	can_interrupt_flag_clear
函数原型	void can_interrupt_flag_clear(uint32_t can_periph, can_interrupt_flag_enum flag);
功能描述	清除CAN中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
can_periph	CAN 外设
<i>CANx(x=0,1)</i>	CAN外设选择, CAN1仅适用于GD32F30X_CL
输入参数{in}	
flag	CAN中断标志位
<i>CAN_INT_FLAG_S</i> <i>LPIF</i>	进入睡眠工作模式的状态改变中断标志
<i>CAN_INT_FLAG_W</i> <i>UIF</i>	从睡眠工作模式唤醒的状态改变中断标志
<i>CAN_INT_FLAG_E</i>	错误中断标志

<i>RRIF</i>	
<i>CAN_INT_FLAG_M</i> <i>TF2</i>	邮箱2发送完成中断标志
<i>CAN_INT_FLAG_M</i> <i>TF1</i>	邮箱1发送完成中断标志
<i>CAN_INT_FLAG_M</i> <i>TF0</i>	邮箱0发送完成中断标志
<i>CAN_INT_FLAG_R</i> <i>F00</i>	接收FIFO0溢出中断标志
<i>CAN_INT_FLAG_R</i> <i>FF0</i>	接收FIFO0满中断标志
<i>CAN_INT_FLAG_R</i> <i>F01</i>	接收FIFO1溢出中断标志
<i>CAN_INT_FLAG_R</i> <i>FF1</i>	接收FIFO1满中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CAN0 mailbox 0 transmit finished interrupt flag */
can_interrupt_flag_clear (CAN0, CAN_INT_FLAG_MTF0);
```

3.5. CRC

循环冗余校验码是一种用在数字网络和存储设备上的差错校验码，可以校验原始数据的偶然误差。章节[3.5.1](#)描述了CRC的寄存器列表，章节[3.5.2](#)对CRC库函数进行说明。

3.5.1. 外设寄存器说明

CRC寄存器列表如下表所示：

表 3-94. CRC 寄存器

寄存器名称	寄存器描述
CRC_DATA	CRC数据寄存器
CRC_FDATA	CRC独立数据寄存器
CRC_CTL	CRC控制寄存器

3.5.2. 外设库函数说明

CRC库函数列表如下表所示：

表 3-95. CRC 库函数

库函数名称	库函数描述
crc_deinit	复位CRC计算单元
crc_data_register_reset	复位数据寄存器，复位后的值为0xFFFFFFFF
crc_data_register_read	读数据寄存器
crc_free_data_register_read	读独立数据寄存器
crc_free_data_register_write	写独立数据寄存器
crc_single_data_calculate	CRC计算一个32位数据
crc_block_data_calculate	CRC计算一个32位数组

函数 `crc_deinit`

函数crc_deinit描述见下表：

表 3-96. 函数 `crc_deinit`

函数名称	crc_deinit

函数原形	void crc_deinit(void);
功能描述	复位CRC计算单元
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset crc */

crc_deinit();
```

函数 **crc_data_register_reset**

函数crc_data_register_reset描述见下表：

表 3-97. 函数 **crc_data_register_reset**

函数名称	crc_data_register_reset
函数原形	void crc_data_register_reset(void);
功能描述	复位数据寄存器，复位后的值为0xFFFFFFFF
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset crc data register */
crc_data_register_reset();
```

函数 **crc_data_register_read**

函数crc_data_register_read描述见下表:

表 3-98. 函数 **crc_data_register_read**

函数名称	crc_data_register_read	
函数原形	uint32_t crc_data_register_read(void);	
功能描述	读数据寄存器	
先决条件	-	
被调用函数	-	
输入参数{in}		
-	-	
输出参数{out}		
-	-	
返回值		
uint32_t	从数据寄存器读取的32位数据 (0-0xFFFFFFFF)	

例如:

```
/* read crc data register */
uint32_t crc_value = 0;
crc_value = crc_data_register_read();
```

函数 **crc_free_data_register_read**

函数crc_free_data_register_read描述见下表:

表 3-99. 函数 **crc_free_data_register_read**

函数名称	crc_free_data_register_read	
函数原形	uint8_t crc_free_data_register_read(void);	
功能描述	读独立数据寄存器	
先决条件	-	
被调用函数	-	

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	从独立数据寄存器读取的8位数据 (0-0xFF)

例如：

```
/* read crc free data register */

uint8_t crc_value = 0;

crc_value = crc_free_data_register_read();
```

函数 **crc_free_data_register_write**

函数crc_free_data_register_write描述见下表：

表 3-100. 函数 **crc_free_data_register_write**

函数名称	crc_free_data_register_write
函数原形	void crc_free_data_register_write(uint8_t free_data);
功能描述	写独立数据寄存器
先决条件	-
被调用函数	-
输入参数{in}	
free_data	设定的8位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write the free data register */

crc_free_data_register_write(0x11);
```

函数 `crc_single_data_calculate`

函数`crc_single_data_calculate`描述见下表：

表 3-101. 函数 `crc_single_data_calculate`

函数名称	<code>crc_single_data_calculate</code>
函数原形	<code>uint32_t crc_single_data_calculate(uint32_t sdata);</code>
功能描述	CRC计算一个32位数据
先决条件	-
被调用函数	-
输入参数{in}	
<code>sdata</code>	设定的32位数据
输出参数{out}	
-	-
返回值	
<code>uint32_t</code>	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

```

/* CRC calculate a 32-bit data */

uint32_t val = 0, valcrc = 0;

val = (uint32_t) 0xabcd1234;

valcrc = crc_single_data_calculate(val);
  
```

函数 `crc_block_data_calculate`

函数`crc_block_data_calculate`描述见下表：

表 3-102. 函数 `crc_block_data_calculate`

函数名称	<code>crc_block_data_calculate</code>
函数原形	<code>uint32_t crc_block_data_calculate(uint32_t array[], uint32_t size);</code>
功能描述	CRC计算一个32位数组
先决条件	-
被调用函数	-
输入参数{in}	

array	32位数据数组的指针
输入参数{in}	
size	数据长度
输出参数{out}	
-	-
返回值	
uint32_t	32位CRC计算结果 (0-0xFFFFFFFF)

例如：

```
/* CRC calculate a 32-bit data array */

#define BUFFER_SIZE    6

uint32_t valcrc = 0;

static const uint32_t data_buffer[BUFFER_SIZE] = {
    0x00001111, 0x00002222, 0x00003333, 0x00004444, 0x00005555, 0x00006666};

valcrc = crc_block_data_calculate((uint32_t *) data_buffer, BUFFER_SIZE);
```

3.6. CTC

CTC模块基于外部高精度的参考信号源来校准IRC48M的时钟频率，通过自动的或手动的调整校准值，以得到一个精准的IRC48M时钟。章节[3.6.1](#)描述了CTC的寄存器列表，章节[3.6.2](#)对CTC库函数进行说明。

3.6.1. 外设寄存器说明

CTC寄存器列表如下表所示：

表 3-103. CTC 寄存器

寄存器名称	寄存器描述
CTC_CTL0	CTC控制寄存器0
CTC_CTL1	CTC控制寄存器1
CTC_STAT	CTC状态寄存器
CTC_INTC	CTC中断清除寄存器

3.6.2. 外设库函数说明

CTC库函数列表如下表所示：

表 3-104. CTC 库函数

库函数名称	库函数描述
ctc_deinit	复位CTC单元
ctc_counter_enable	使能CTC校准
ctc_counter_disable	禁能CTC校准
ctc_irc48m_trim_value_config	配置IRC48M时钟校准值
ctc_software_refsource_pulse_generate	产生CTC参考时钟源同步脉冲
te	
ctc_hardware_trim_mode_config	CTC硬件自动校准模式配置
ctc_refsource_polarity_config	CTC参考信号源时钟极性配置
ctc_refsource_signal_select	CTC参考信号源选择
ctc_refsource_prescaler_config	CTC参考信号源分频配置
ctc_clock_limit_value_config	CTC时钟校准时基限值设置
ctc_counter_reload_value_config	CTC计数器重载值配置
ctc_counter_capture_value_read	读取CTC计数器捕获值
ctc_counter_direction_read	读取CTC校准时钟计数方向
ctc_counter_reload_value_read	读取CTC计数器重载值
ctc_irc48m_trim_value_read	读取IRC48M校准值
ctc_interrupt_enable	CTC中断使能
ctc_interrupt_disable	CTC中断禁能
ctc_interrupt_flag_get	CTC中断标志获取
ctc_interrupt_flag_clear	CTC中断标志清除
ctc_flag_get	CTC状态标志获取
ctc_flag_clear	CTC状态标志清除

函数 `ctc_deinit`

函数`ctc_deinit`描述见下表：

表 3-105. 函数 ctc_deinit

函数名称	ctc_deinit
函数原形	void ctc_deinit (void);
功能描述	复位CTC单元
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset CTC */
ctc_deinit();
```

函数 ctc_counter_enable

函数ctc_counter_enable描述见下表：

表 3-106. 函数 ctc_counter_enable

函数名称	ctc_counter_enable
函数原形	void ctc_counter_enable (void);
功能描述	使能CTC校准计数器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* enable CTC trim counter*/
ctc_counter_enable();
```

函数 **ctc_counter_disable**

函数ctc_counter_disable描述见下表：

表 3-107. 函数 **ctc_counter_disable**

函数名称	ctc_counter_disable
函数原形	void ctc_counter_disable (void);
功能描述	禁能CTC计数器校准
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable CTC trim counter */
ctc_counter_disable();
```

函数 **ctc_irc48m_trim_value_config**

函数ctc_irc48m_trim_value_config描述见下表：

表 3-108. 函数 **ctc_irc48m_trim_value_config**

函数名称	ctc_irc48m_trim_value_config
函数原形	void ctc_irc48m_trim_value_config(uint8_t trim_value);
功能描述	配置IRC48M时钟校准值
先决条件	-

被调用函数	-
输入参数{in}	
trim_value	0~63
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* IRC48M trim value configuration */

ctc_irc48m_trim_value_config (0x01);
```

函数 **ctc_software_refsource_pulse_generate**

函数 **ctc_software_refsource_pulse_generate** 描述见下表：

表 3-109. 函数 **ctc_software_refsource_pulse_generate**

函数名称	ctc_software_refsource_pulse_generate
函数原形	void ctc_software_refsource_pulse_generate(void);
功能描述	产生CTC参考时钟源同步脉冲
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* generate reference source sync pulse */

ctc_software_refsource_pulse_generate ();
```

函数 `ctc.hardware.trim_mode_config`

函数`ctc.hardware.trim_mode_config`描述见下表:

表 3-110. 函数 `ctc.hardware.trim_mode_config`

函数名称	<code>ctc.hardware.trim_mode_config</code>
函数原形	<code>void ctc.hardware.trim_mode_config(uint32_t hardmode);</code>
功能描述	配置硬件自动校准
先决条件	-
被调用函数	-
输入参数{in}	
hardmode	硬件校准开启还是关闭
<code>CTC_HARDWARE_TRIM_MODE_ENA_BLE</code>	硬件校准开启
<code>CTC_HARDWARE_TRIM_MODE_DISA_BLE</code>	硬件校准关闭
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable CTC hardware trim */

ctc.hardware.trim_mode_config (CTC_HARDWARE_TRIM_MODE_ENABLE);
```

函数 `ctc.refsource_polarity_config`

函数`ctc.refsource_polarity_config`描述见下表:

表 3-111. 函数 `ctc.refsource_polarity_config`

函数名称	<code>ctc.refsource_polarity_config</code>
函数原形	<code>void ctc.refsource_polarity_config(uint32_t polarity);</code>
功能描述	CTC参考时钟极性配置
先决条件	-

被调用函数	-
输入参数{in}	
polarity	时钟极性
CTC_REFRESOURCE_POLARITY_FALLING	参考信号源的同步极性为下降沿
CTC_REFRESOURCE_POLARITY_RISING	参考信号源的同步极性为上升沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set reference source polarity */
ctc_refresource_polarity_config (CTC_REFRESOURCE_POLARITY_RISING);
```

函数 ctc_refresource_signal_select

函数ctc_refresource_signal_select描述见下表：

表 3-112. 函数 ctc_refresource_signal_select

函数名称	ctc_refresource_signal_select
函数原形	void ctc_refresource_signal_select(uint32_t refs);
功能描述	CTC参考信号源选择
先决条件	-
被调用函数	-
输入参数{in}	
refs	参考信号源
CTC_REFRESOURCE_GPIO	选择GPIO输入信号
CTC_REFRESOURCE_LXTAL	选择LXTAL时钟

<code>CTC_REFRESOURCE_USBSOF</code>	选择USBD_SOF 或者 USBFS_SOF信号
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reference signal selection */

ctc_refresource_signal_select (CTC_REFRESOURCE_LXTAL);
```

函数 `ctc_refresource_prescaler_config`

函数`ctc_refresource_prescaler_config`描述见下表：

表 3-113. 函数 `ctc_refresource_prescaler_config`

函数名称	<code>ctc_refresource_prescaler_config</code>
函数原形	<code>void ctc_refresource_prescaler_config(uint32_t prescaler);</code>
功能描述	参考信号源的分频设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>prescaler</code>	分频系数
<code>CTC_REFRESOURCE_PSC_OFF</code>	参考信号不分频
<code>CTC_REFRESOURCE_PSC_DIV2</code>	参考信号2分频
<code>CTC_REFRESOURCE_PSC_DIV4</code>	参考信号4分频
<code>CTC_REFRESOURCE_PSC_DIV8</code>	参考信号8分频
<code>CTC_REFRESOURCE_PSC_DIV16</code>	参考信号16分频
<code>CTC_REFRESOURCE</code>	参考信号32分频

<code>_PSC_DIV32</code>	
<code>CTC_REFRESOURCE_PSC_DIV64</code>	参考信号64分频
<code>CTC_REFRESOURCE_PSC_DIV128</code>	参考信号128分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure reference signal source prescaler */
ctc_refresource_prescaler_config(CTC_REFRESOURCE_PSC_DIV2);
```

函数 `ctc_clock_limit_value_config`

函数`ctc_clock_limit_value_config`描述见下表：

表 3-114. 函数 `ctc_clock_limit_value_config`

函数名称	<code>ctc_clock_limit_value_config</code>
函数原形	<code>void ctc_clock_limit_value_config(uint8_t limit_value);</code>
功能描述	CTC时钟校准时基限值设置
先决条件	-
被调用函数	-
输入参数{in}	
<code>limit_value</code>	0x00 - 0xFF
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure clock trim base limit value */
ctc_clock_limit_value_config (0x1F);
```

函数 ctc_counter_reload_value_config

函数ctc_counter_reload_value_config描述见下表:

表 3-115. 函数 ctc_counter_reload_value_config

函数名称	ctc_counter_reload_value_config
函数原形	void ctc_counter_reload_value_config(uint16_t reload_value);
功能描述	CTC计数器重载值设置
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	0x0000 - 0xFFFF
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CTC counter reload value */
ctc_counter_reload_value_config (0x00FF);
```

函数 ctc_counter_capture_value_read

函数ctc_counter_capture_value_read描述见下表:

表 3-116. 函数 ctc_counter_capture_value_read

函数名称	ctc_counter_capture_value_read
函数原形	uint16_t ctc_counter_capture_value_read(void);
功能描述	读取计数器捕获值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
uint16_t	读取计数器捕获值(0x0000 - 0xFFFF)

例如：

```
/* read CTC counter capture value */

uint16_t ctc_value = 0;

ctc_value = ctc_counter_capture_value_read();
```

函数 **ctc_counter_direction_read**

函数 **ctc_counter_direction_read** 描述见下表：

表 3-117. 函数 **ctc_counter_direction_read**

函数名称	ctc_counter_direction_read
函数原形	FlagStatus ctc_counter_direction_read(void);
功能描述	读取CTC校准时钟计数方向
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET(向下计数) / RESET(向上计数)

例如：

```
/* read ctc counter direction */

FlagStatus ctc_direction = SET;

ctc_direction = ctc_counter_direction_read();
```

函数 **ctc_counter_reload_value_read**

函数 **ctc_counter_reload_value_read** 描述见下表：

表 3-118. 函数 ctc_counter_reload_value_read

函数名称	ctc_counter_reload_value_read
函数原形	uint16_t ctc_counter_reload_value_read(void);
功能描述	读取CTC计数器重载值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint16_t	读取计数器重载值的16位数据 (0x0000 - 0xFFFF)

例如：

```
/* read CTC counter reload value */
uint16_t ctc_reload_value = 0;
ctc_reload_value = ctc_counter_reload_value_read();
```

函数 ctc_irc48m_trim_value_read

函数ctc_irc48m_trim_value_read描述见下表：

表 3-119. 函数 ctc_irc48m_trim_value_read

函数名称	ctc_irc48m_trim_value_read
函数原形	uint8_t ctc_irc48m_trim_value_read(void);
功能描述	读IRC48M校准值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint8_t	6位IRC48M校准值 (0-63)

例如：

```
/* read the IRC48M trim value */

uint8_t ctc_trim_value = 0;

ctc_trim_value = ctc_irc48m_trim_value_read();
```

函数 **ctc_interrupt_enable**

函数**ctc_interrupt_enable**描述见下表：

表 3-120. 函数 **ctc_interrupt_enable**

函数名称	ctc_interrupt_enable
函数原形	void ctc_interrupt_enable(uint32_t interrupt);
功能描述	使能外设CTC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	CTC中断
CTC_INT_CKOK	时钟校准完成中断
CTC_INT_CKWAR N	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREF	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable CTC clock trim OK interrupt */

ctc_interrupt_enable (CTC_INT_CKOK);
```

函数 ctc_interrupt_disable

函数ctc_interrupt_disable描述见下表:

表 3-121. 函数 ctc_interrupt_disable

函数名称	ctc_interrupt_disable
函数原形	void ctc_interrupt_disable(uint32_t interrupt);
功能描述	禁能外设CTC中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	CTC中断
CTC_INT_CKOK	时钟校准完成中断
CTC_INT_CKWAR N	时钟校准警告中断
CTC_INT_ERR	错误中断
CTC_INT_EREF	期望参考信号中断
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable CTC clock trim OK interrupt */

ctc_interrupt_disable (CTC_INT_CKOK);
```

函数 ctc_interrupt_flag_get

函数ctc_interrupt_flag_get描述见下表:

表 3-122. 函数 ctc_interrupt_flag_get

函数名称	ctc_interrupt_flag_get
函数原形	FlagStatus ctc_interrupt_flag_get(uint32_t int_flag);
功能描述	获取CTC中断标志位

先决条件	-
被调用函数	-
输入参数{in}	
int_flag	CTC中断标志
<i>CTC_INT_FLAG_C_KOK</i>	时钟校准完成中断标志位
<i>CTC_INT_FLAG_C_KWARN</i>	时钟校准警告中断标志位
<i>CTC_INT_FLAG_E_RR</i>	错误中断标志位
<i>CTC_INT_FLAG_E_REF</i>	期望参考信号中断标志位
<i>CTC_INT_FLAG_C_KERR</i>	时钟校准错误位
<i>CTC_INT_FLAG_R_EFMISS</i>	参考同步脉冲信号丢失
<i>CTC_INT_FLAG_T_RIMERR</i>	校准值错误位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get CTC interrupt flag status */
FlagStatus state = ctc_interrupt_flag_get (CTC_INT_FLAG_CKOK);
```

函数 **ctc_interrupt_flag_clear**

函数**ctc_interrupt_flag_clear**描述见下表：

表 3-123. 函数 **ctc_interrupt_flag_clear**

函数名称	ctc_interrupt_flag_clear
函数原形	void ctc_interrupt_flag_clear(uint32_t int_flag);

功能描述	清除CTC中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
<code>int_flag</code>	CTC中断标志
<code>CTC_INT_FLAG_C_KOK</code>	时钟校准完成中断标志位
<code>CTC_INT_FLAG_C_KWARN</code>	时钟校准警告中断标志位
<code>CTC_INT_FLAG_E_RR</code>	错误中断标志位
<code>CTC_INT_FLAG_E_REF</code>	期望参考信号中断标志位
<code>CTC_INT_FLAG_C_KERR</code>	时钟校准错误位
<code>CTC_INT_FLAG_R_EFMISS</code>	参考同步脉冲信号丢失
<code>CTC_INT_FLAG_T_RIMERR</code>	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*clear CTC interrupt flag status */
ctc_interrupt_flag_clear (CTC_INT_FLAG_CKOK);
```

函数 `ctc_flag_get`

函数`ctc_flag_get`描述见下表：

表 3-124. 函数 `ctc_flag_get`

函数名称	<code>ctc_flag_get</code>
------	---------------------------

函数原形	FlagStatus ctc_flag_get (uint32_t flag);
功能描述	获取CTC状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CTC状态标志
CTC_FLAG_CKOK	时钟校准完成标志位
CTC_FLAG_CKWA RN	时钟校准警告中断标志位
CTC_FLAG_ERR	错误中断标志位
CTC_FLAG_EREF	期望参考信号中断标志位
CTC_FLAG_CKER R	时钟校准错误位
CTC_FLAG_REFMI SS	参考同步脉冲信号丢失
CTC_FLAG_TRIME RR	校准值错误位
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get CTC flag status */
FlagStatus state = ctc_flag_get (CTC_FLAG_CKOK);
```

函数 **ctc_flag_clear**

函数**ctc_flag_clear**描述见下表：

表 3-125. 函数 **ctc_flag_clear**

函数名称	ctc_flag_clear
函数原形	void ctc_flag_clear (uint32_t flag);

功能描述	清除CTC状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag	CTC状态标志
CTC_FLAG_CKOK	时钟校准完成标志位
CTC_FLAG_CKWA RN	时钟校准警告中断标志位
CTC_FLAG_ERR	错误中断标志位
CTC_FLAG_EREF	期望参考信号中断标志位
CTC_FLAG_CKER R	时钟校准错误位
CTC_FLAG_REFMI SS	参考同步脉冲信号丢失
CTC_FLAG_TRIME RR	校准值错误位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear CTC flag status */
ctc_flag_clear (CTC_FLAG_CKOK);
```

3.7. DAC

数字/模拟转换器可以将12位的数字数据转换为外部引脚上的电压输出，章节[3.7.1](#)描述了DAC的寄存器列表，章节[3.7.2](#)对DAC库函数进行说明。

3.7.1. 外设寄存器说明

DAC寄存器列表如下表所示：

表 3-126. DAC 寄存器

寄存器名称	寄存器描述
DAC_CTL	DAC控制寄存器
DAC_SWT	DAC软件触发寄存器
DAC0_R12DH	DAC0 12位右对齐数据保持寄存器
DAC0_L12DH	DAC0 12位左对齐数据保持寄存器
DAC0_R8DH	DAC0 8位右对齐数据保持寄存器
DAC1_R12DH	DAC1 12位右对齐数据保持寄存器
DAC1_L12DH	DAC1 12位左对齐数据保持寄存器
DAC1_R8DH	DAC1 8位右对齐数据保持寄存器
DACC_R12DH	DAC 并发模式12位右对齐数据保持寄存器
DACC_L12DH	DAC并发模式12位左对齐数据保持寄存器
DACC_R8DH	DAC并发模式8位右对齐数据保持寄存器
DAC0_DO	DAC0数据输出寄存器
DAC1_DO	DAC1数据输出寄存器

3.7.2. 外设库函数说明

DAC库函数列表如下表所示:

表 3-127. DAC 库函数

库函数名称	库函数描述
dac_deinit	DAC外设复位
dac_enable	DAC使能
dac_disable	DAC禁能
dac_dma_enable	DAC的DMA功能使能
dac_dma_disable	DAC的DMA功能禁能
dac_output_buffer_enable	DAC输出缓冲区使能
dac_output_buffer_disable	DAC输出缓冲区禁能
dac_output_value_get	DAC输出数据获取
dac_data_set	DAC输出数据设置

库函数名称	库函数描述
dac_trigger_enable	DAC触发使能
dac_trigger_disable	DAC触发禁能
dac_trigger_source_config	DAC触发源选择
dac_software_trigger_enable	DAC软件触发使能
dac_software_trigger_disable	DAC软件触发禁能
dac_wave_mode_config	DAC噪声波模式选择
dac_wave_bit_width_config	DAC噪声波位宽设置
dac_lfsr_noise_config	DAC LFSR模式设置
dac_triangle_noise_config	DAC三角波模式设置
dac_concurrent_enable	并发DAC模式使能
dac_concurrent_disable	并发DAC模式禁能
dac_concurrent_software_trigger_enable	并发DAC模式软件触发使能
dac_concurrent_software_trigger_disable	并发DAC模式软件触发禁能
dac_concurrent_output_buffer_enable	并发DAC模式输出缓冲区使能
dac_concurrent_output_buffer_disable	并发DAC模式输出缓冲区禁能
dac_concurrent_data_set	并发DAC模式输出数据设置

dac_deinit

函数dac_deinit描述见下表：

表 3-128. 函数 dac_deinit

函数名称	dac_deinit
函数原型	void dac_deinit(void);
功能描述	DAC外设复位
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize DAC */
dac_deinit();
```

dac_enable

函数dac_enable描述见下表:

表 3-129. 函数 **dac_enable**

函数名称	dac_enable
函数原型	void dac_enable(uint32_t dac_periph);
功能描述	DAC使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 */
dac_enable(DAC0);
```

dac_disable

函数dac_disable描述见下表:

表 3-130. 函数 **dac_disable**

函数名称	dac_disable
函数原型	void dac_disable(uint32_t dac_periph);
功能描述	DAC禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 */
dac_disable(DAC0);
```

dac_dma_enable

函数dac_dma_enable描述见下表:

 表 3-131. 函数 **dac_dma_enable**

函数名称	dac_dma_enable
函数原型	void dac_dma_enable(uint32_t dac_periph);
功能描述	DAC的DMA功能使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable DAC0 DMA function */
dac_dma_enable(DAC0);
```

dac_dma_disable

函数dac_dma_disable描述见下表：

表 3-132. 函数 dac_dma_disable

函数名称	dac_dma_disable
函数原型	void dac_dma_disable(uint32_t dac_periph);
功能描述	DAC的DMA功能禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0 DMA function */
dac_dma_disable(DAC0);
```

dac_output_buffer_enable

函数dac_output_buffer_enable描述见下表：

表 3-133. 函数 dac_output_buffer_enable

函数名称	dac_output_buffer_enable
------	--------------------------

函数原型	void dac_output_buffer_enable(uint32_t dac_periph);
功能描述	DAC输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC0 output buffer */
dac_output_buffer_enable(DAC0);
```

dac_output_buffer_disable

函数dac_output_buffer_disable描述见下表:

表 3-134. 函数 **dac_output_buffer_disable**

函数名称	dac_output_buffer_disable
函数原型	void dac_output_buffer_disable(uint32_t dac_periph);
功能描述	DAC输出缓冲区禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable DAC0 output buffer */

dac_output_buffer_disable(DAC0);
```

dac_output_value_get

函数dac_output_value_get描述见下表：

表 3-135. 函数 dac_output_value_get

函数名称	dac_output_value_get
函数原型	uint16_t dac_output_value_get(uint32_t dac_periph);
功能描述	DAC输出数据获取
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输出参数{out}	
-	DAC输出数据
返回值	
uint16_t	外设DACx数据保持寄存器值 (0x000~0x7FF)

例如：

```
/* get the last data output value */

data = dac_output_value_get(DAC0);
```

dac_data_set

函数dac_data_set描述见下表：

表 3-136. 函数 dac_data_set

函数名称	dac_data_set
函数原型	void dac_data_set(uint32_t dac_periph, uint32_t dac_align, uint16_t data);
功能描述	DAC输出数据设置

先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输入参数{in}	
dac_align	DAC对齐模式
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
输入参数{in}	
data	写入DACx的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC0 data holding register value */
dac_data_set(DAC0, DAC_ALIGN_8B_R, 0xff);
```

dac_trigger_enable

函数**dac_trigger_enable**描述见下表:

表 3-137. 函数 dac_trigger_enable

函数名称	dac_trigger_enable
函数原型	void dac_trigger_enable(uint32_t dac_periph);
功能描述	DAC触发使能
先决条件	-
被调用函数	-

输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0 trigger */
dac_trigger_enable(DAC0);
```

dac_trigger_disable

函数dac_trigger_disable描述见下表：

表 3-138. 函数 **dac_trigger_disable**

函数名称	dac_trigger_disable
函数原型	void dac_trigger_disable(uint32_t dac_periph);
功能描述	DAC触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC0 trigger */
dac_trigger_disable(DAC0);
```

dac_trigger_source_config

函数dac_trigger_source_config描述见下表:

表 3-139. 函数 dac_trigger_source_config

函数名称	dac_trigger_source_config
函数原型	void dac_trigger_source_config(uint32_t dac_periph,uint32_t triggersource);
功能描述	DAC触发源选择
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输入参数{in}	
triggersource	DAC触发源
DAC_TRIGGER_T1_TRGO	TIMER1 TRGO
DAC_TRIGGER_T2_TRGO	TIMER2 TRGO (互联型)
DAC_TRIGGER_T3_TRGO	TIMER3 TRGO
DAC_TRIGGER_T4_TRGO	TIMER4 TRGO
DAC_TRIGGER_T5_TRGO	TIMER5 TRGO
DAC_TRIGGER_T6_TRGO	TIMER6 TRGO
DAC_TRIGGER_T7_TRGO	TIMER7 TRGO (非互联型)
DAC_TRIGGER_EXTI_9	EXTI线9中断
DAC_TRIGGER_SOFTWARE	软件触发
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure DAC0 trigger source */
dac_trigger_source_config(DAC0, DAC_TRIGGER_T1_TRGO);
```

dac_software_trigger_enable

函数dac_software_trigger_enable描述见下表：

表 3-140. 函数 **dac_software_trigger_enable**

函数名称	dac_software_trigger_enable
函数原型	void dac_software_trigger_enable(uint32_t dac_periph);
功能描述	DAC软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DAC0 software trigger */
dac_software_trigger_enable(DAC0);
```

dac_software_trigger_disable

函数dac_software_trigger_disable描述见下表：

表 3-141. 函数 **dac_software_trigger_disable**

函数名称	dac_software_trigger_disable
------	------------------------------

函数原型	void dac_software_trigger_disable(uint32_t dac_periph);
功能描述	DAC软件触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC0 software trigger */
dac_software_trigger_disable(DAC0);
```

dac_wave_mode_config

函数dac_wave_mode_config描述见下表:

表 3-142. 函数 **dac_wave_mode_config**

函数名称	dac_wave_mode_config
函数原型	void dac_wave_mode_config(uint32_t dac_periph, uint32_t wave_mode);
功能描述	DAC噪声波模式选择
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输入参数{in}	
wave_mode	噪声波模式选择
DAC_WAVE_	噪声波模式禁能

<i>DISABLE</i>	
<i>DAC_WAVE_MODE_LFSR</i>	LFSR噪声波模式
<i>DAC_WAVE_MODE_TRIANGLE</i>	三角波噪声波模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0 wave mode */
dac_wave_mode_config(DAC0, DAC_WAVE_DISABLE);
```

dac_wave_bit_width_config

函数dac_wave_bit_width_config描述见下表:

表 3-143. 函数 **dac_wave_bit_width_config**

函数名称	dac_wave_bit_width_config
函数原型	void dac_wave_bit_width_config(uint32_t dac_periph, uint32_t bit_width);
功能描述	DAC噪声波位宽设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
<i>DACx</i>	DAC外设选择 (<i>x</i> =0,1)
输入参数{in}	
bit_width	噪声波位宽
<i>DAC_WAVE_BIT_WIDTH_x</i>	噪声波位宽 (<i>x</i> = 1..12)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure DAC0 wave bit width */
dac_wave_bit_width_config(DAC0,DAC_WAVE_BIT_WIDTH_1);
```

dac_lfsr_noise_config

函数dac_lfsr_noise_config描述见下表：

表 3-144. 函数 dac_lfsr_noise_config

函数名称	dac_lfsr_noise_config
函数原型	void dac_lfsr_noise_config(uint32_t dac_periph, uint32_t unmask_bits);
功能描述	DAC LFSR模式设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输入参数{in}	
unmask_bits	噪声波的非屏蔽位宽
DAC_LFSR_BIT0	LFSR模式位0非屏蔽
DAC_LFSR_BITSx_0	LFSR模式位[x:0]非屏蔽
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure DAC0 LFSR noise mode */
dac_lfsr_noise_config(DAC0, DAC_LFSR_BIT0);
```

dac_triangle_noise_config

函数dac_triangle_noise_config描述见下表:

表 3-145. 函数 dac_triangle_noise_config

函数名称	dac_triangle_noise_config
函数原型	void dac_triangle_noise_config(uint32_t dac_periph, uint32_t amplitude);
功能描述	DAC三角波模式设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_periph	DAC外设
DACx	DAC外设选择 (x =0,1)
输入参数{in}	
amplitude	三角波幅值
DAC_TRIANGLE_AMPLITUDE_x	x = 2 ⁿ -1 (n = 1..12)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure DAC0 triangle noise mode */
dac_triangle_noise_config(DAC0, DAC_TRIANGLE_AMPLITUDE_1);
```

dac_concurrent_enable

函数dac_concurrent_enable描述见下表:

表 3-146. 函数 dac_concurrent_enable

函数名称	dac_concurrent_enable
函数原型	void dac_concurrent_enable (void);
功能描述	并发DAC模式使能

先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC concurrent mode */
```

```
dac_concurrent_enable();
```

dac_concurrent_disable

函数dac_concurrent_disable描述见下表:

表 3-147. 函数 dac_concurrent_disable

函数名称	dac_concurrent_disable
函数原型	void dac_concurrent_disable (void);
功能描述	并发DAC模式禁能
先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DAC concurrent mode */
```

```
dac_concurrent_disable();
```

dac_concurrent_software_trigger_enable

函数dac_concurrent_software_trigger_enable描述见下表:

表 3-148. 函数 **dac_concurrent_software_trigger_enable**

函数名称	dac_concurrent_software_trigger_enable
函数原型	void dac_concurrent_software_trigger_enable(void);
功能描述	并发DAC模式软件触发使能
先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC concurrent software trigger */
dac_concurrent_software_trigger_enable();
```

dac_concurrent_software_trigger_disable

函数dac_concurrent_software_trigger_disable描述见下表:

表 3-149. 函数 **dac_concurrent_software_trigger_disable**

函数名称	dac_concurrent_software_trigger_disable
函数原型	void dac_concurrent_software_trigger_disable(void);
功能描述	并发DAC模式软件触发禁能
先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	

-	-
返回值	
-	-

例如:

```
/* disable DAC concurrent software trigger */

dac_concurrent_software_trigger_disable();
```

dac_concurrent_output_buffer_enable

函数dac_concurrent_output_buffer_enable描述见下表:

表 3-150. 函数 dac_concurrent_output_buffer_enable

函数名称	dac_concurrent_output_buffer_enable
函数原型	void dac_concurrent_output_buffer_enable(void);
功能描述	并发DAC模式输出缓冲区使能
先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable DAC concurrent buffer function */

dac_concurrent_output_buffer_enable();
```

dac_concurrent_output_buffer_disable

函数dac_concurrent_output_buffer_disable描述见下表:

表 3-151. 函数 dac_concurrent_output_buffer_disable

函数名称	dac_concurrent_output_buffer_disable
函数原型	void dac_concurrent_output_buffer_disable(void);

功能描述	并发DAC模式输出缓冲区禁能
先决条件	-
被调用函数	-
输入参数{in}	
	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DAC concurrent buffer function */

dac_concurrent_output_buffer_disable();
```

dac_concurrent_data_set

函数dac_concurrent_data_set描述见下表：

表 3-152. 函数 **dac_concurrent_data_set**

函数名称	dac_concurrent_data_set
函数原型	void dac_concurrent_data_set(uint32_t dac_align, uint16_t data0, uint16_t data1);
功能描述	并发DAC模式输出数据设置
先决条件	-
被调用函数	-
输入参数{in}	
dac_align	DAC对齐模式
<i>DAC_ALIGN_8B_R</i>	8位数据右对齐
<i>DAC_ALIGN_12B_R</i>	12位数据右对齐
<i>DAC_ALIGN_12B_L</i>	12位数据左对齐
输入参数{in}	
data0	写入DAC0的数据

输入参数{in}	
data1	写入DAC1的数据
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set DAC concurrent mode data holding register value */
dac_concurrent_data_set(DAC_ALIGN_8B_R, 0xff, 0xff);
```

3.8. DBG

调试系统帮助调试者在低功耗模式下调试或者进行一些外设调试。章节[3.8.1](#)描述了DBG的寄存器列表，章节[3.8.2](#)对DBG库函数进行说明。

3.8.1. 外设寄存器说明

DBG寄存器列表如下表所示:

表 3-153. DBG 寄存器

寄存器名称	寄存器描述
DBG_ID	DBG ID寄存器
DBG_CTL0	DBG控制寄存器0

3.8.2. 外设库函数说明

DBG库函数列表如下表所示:

表 3-154. DBG 库函数

库函数名称	库函数描述
dbg_deinit	复位DEBUG寄存器
dbg_id_get	读DBG_ID寄存器
dbg_low_power_enable	使能低功耗模式的MCU调试保持功能
dbg_low_power_disable	禁能低功耗模式的MCU调试保持功能

库函数名称	库函数描述
dbg_periph_enable	使能外设的MCU调试保持功能
dbg_periph_disable	禁能外设的MCU调试保持功能
dbg_trace_pin_enable	使能跟踪引脚分配
dbg_trace_pin_disable	禁能跟踪引脚分配
dbg_trace_pin_mode_set	配置跟踪引脚分配模式

枚举类型 `dbg_periph_enum`

表 3-155. 枚举类型 `dbg_periph_enum`

成员名称	功能描述
DBG_FWDGT_HOLD	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD	当内核停止时，保持WWDT计数器时钟
DBG_TIMER0_HOLD	当内核停止时，保持TIMER0计数器计数值不变
DBG_TIMER1_HOLD	当内核停止时，保持TIMER1计数器计数值不变
DBG_TIMER2_HOLD	当内核停止时，保持TIMER2计数器计数值不变
DBG_TIMER3_HOLD	当内核停止时，保持TIMER3计数器计数值不变
DBG_CAN0_HOLD	当内核停止时，CAN0接收寄存器停止接收数据
DBG_I2C0_HOLD	当内核停止时，保持I2C0的SMBUS状态不变，用于调试
DBG_I2C1_HOLD	当内核停止时，保持I2C1的SMBUS状态不变，用于调试
DBG_TIMER4_HOLD	当内核停止时，保持TIMER4计数器计数值不变
DBG_TIMER5_HOLD	当内核停止时，保持TIMER5计数器计数值不变
DBG_TIMER6_HOLD	当内核停止时，保持TIMER6计数器计数值不变
DBG_TIMER7_HOLD	当内核停止时，保持TIMER7计数器计数值不变
DBG_CAN1_HOLD	当内核停止时，CAN1接收寄存器停止接收数据
DBG_TIMER11_HOLD	当内核停止时，保持TIMER11计数器计数值不变
DBG_TIMER12_HOLD	当内核停止时，保持TIMER12计数器计数值不变
DBG_TIMER13_HOLD	当内核停止时，保持TIMER13计数器计数值不变
DBG_TIMER8_HOLD	当内核停止时，保持TIMER8计数器计数值不变

DBG_TIMER9_HOLD	当内核停止时，保持TIMER9计数器计数值不变
DBG_TIMER10_HOLD	当内核停止时，保持TIMER10计数器计数值不变

函数 **dbg_deinit**

函数dbg_deinit描述见下表：

表 3-156. 函数 **dbg_deinit**

函数名称	dbg_deinit
函数原形	void dbg_deinit(void);
功能描述	复位DEBUG寄存器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	DBG ID (0-0xFFFFFFFF)

例如：

```
/* read DBG_ID code register */

uint32_t id_value = 0;

id_value = dbg_id_get();
```

函数 **dbg_id_get**

函数dbg_id_get描述见下表：

表 3-157. 函数 **dbg_id_get**

函数名称	dbg_id_get
函数原形	uint32_t dbg_id_get(void);
功能描述	读DBG_ID寄存器
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	DBG ID (0-0xFFFFFFFF)

例如：

```
/* read DBG_ID code register */

uint32_t id_value = 0;

id_value = dbg_id_get();
```

函数 **dbg_low_power_enable**

函数dbg_low_power_enable描述见下表：

表 3-158. 函数 **dbg_low_power_enable**

函数名称	dbg_low_power_enable
函数原形	void dbg_low_power_enable(uint32_t dbg_low_power);
功能描述	使能低功耗模式的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_low_power	低功耗模式调试保持
DBG_LOW_POWER_SLEEP	在睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_DEEPSLEEP	在深度睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_STANDBY	在待机模式下，保持调试器连接，可进行调试
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable low power behavior when the mcu is in debug mode */

dbg_low_power_enable(DBG_LOW_POWER_SLEEP);
```

函数 **dbg_low_power_disable**

函数dbg_low_power_disable描述见下表：

表 3-159. 函数 **dbg_low_power_disable**

函数名称	dbg_low_power_disable
函数原形	void dbg_low_power_disable(uint32_t dbg_low_power);
功能描述	禁能低功耗模式的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_low_power	低功耗模式调试保持
DBG_LOW_POWER_R_SLEEP	在睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_R_DEEPSLEEP	在深度睡眠模式下，保持调试器连接，可进行调试
DBG_LOW_POWER_R_STANDBY	在待机模式下，保持调试器连接，可进行调试
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable low power behavior when the mcu is in debug mode */

dbg_low_power_disable(DBG_LOW_POWER_SLEEP);
```

函数 `dbg_periph_enable`

函数`dbg_periph_enable`描述见下表:

表 3-160. 函数 `dbg_periph_enable`

函数名称	<code>dbg_periph_enable</code>
函数原形	<code>void dbg_periph_enable(dbg_periph_enum dbg_periph);</code>
功能描述	使能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
<code>dbg_periph</code>	Peripheral refer to 枚举类型<code>dbg_periph_enum</code>
<code>DBG_FWDGT_HOLD</code>	当内核停止时，保持FWDGT计数器时钟
<code>DBG_WWDGT_HOLD</code>	当内核停止时，保持WWDGTH计数器时钟
<code>DBG_CANx_HOLD</code>	当内核停止时，CANx接收寄存器停止接收数据
<code>DBG_I2Cx_HOLD</code>	当内核停止时，保持I2Cx (x=0,1) 的SMBUS状态不变，用于调试
<code>DBG_TIMERx_HOLD</code>	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable peripheral behavior when the mcu is in debug mode */
dbg_periph_enable(DBG_TIMER0_HOLD);
```

函数 `dbg_periph_disable`

函数`dbg_periph_disable`描述见下表:

表 3-161. 函数 `dbg_periph_disable`

函数名称	<code>dbg_periph_disable</code>
------	---------------------------------

函数原形	void dbg_periph_disable(dbg_periph_enum dbg_periph);
功能描述	禁能外设的MCU调试保持功能
先决条件	-
被调用函数	-
输入参数{in}	
dbg_periph	Peripheral refer to 枚举类型dbg_periph_enum
DBG_FWDGT_HOLD_D	当内核停止时，保持FWDGT计数器时钟
DBG_WWDGT_HOLD_LD	当内核停止时，保持WWDGT计数器时钟
DBG_CANx_HOLD	当内核停止时，CANx接收寄存器停止接收数据（x=0,1）
DBG_I2Cx_HOLD	当内核停止时，保持I2Cx（x=0,1）的SMBUS状态不变，用于调试
DBG_TIMERx_HOLD_D	当内核停止时，保持TIMERx计数器计数值不变 (x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable peripheral behavior when the mcu is in debug mode */

dbg_periph_disable(DBG_TIMER0_HOLD);
```

函数 **dbg_trace_pin_enable**

函数dbg_trace_pin_enable描述见下表：

表 3-162. 函数 **dbg_trace_pin_enable**

函数名称	dbg_trace_pin_enable
函数原形	void dbg_trace_pin_enable(void);
功能描述	使能跟踪引脚分配
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable trace pin assignment */

dbg_trace_pin_enable();
```

函数 **dbg_trace_pin_disable**

函数dbg_trace_pin_disable描述见下表：

表 3-163. 函数 **dbg_trace_pin_disable**

函数名称	dbg_trace_pin_disable
函数原形	void dbg_trace_pin_disable(void);
功能描述	禁能跟踪引脚分配
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable trace pin assignment */

dbg_trace_pin_disable();
```

函数 **dbg_trace_pin_mode_set**

函数dbg_trace_pin_mode_set描述见下表：

表 3-164. 函数 `dbg_trace_pin_mode_set`

函数名称	<code>dbg_trace_pin_mode_set</code>
函数原形	<code>void dbg_trace_pin_mode_set(uint32_t trace_mode);</code>
功能描述	配置跟踪引脚分配模式
先决条件	-
被调用函数	-
输入参数{in}	
<code>trace_mode</code>	跟踪引脚分配模式选择
<code>TRACE_MODE_ASYNC</code>	跟踪引脚用于异步模式
<code>TRACE_MODE_SYN_NC_DATASIZE_1</code>	跟踪引脚用于同步模式且数据长度为1
<code>TRACE_MODE_SYN_NC_DATASIZE_2</code>	跟踪引脚用于同步模式且数据长度为2
<code>TRACE_MODE_SYN_NC_DATASIZE_4</code>	跟踪引脚用于同步模式且数据长度为4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* trace pin mode selection */

dbg_trace_pin_mode_set(TRACE_MODE_ASYNC);
```

3.9. DMA

DMA控制器提供了一种硬件的方式在外设和存储器之间或者存储器和存储器之间传输数据，而无需CPU的介入，从而使CPU可以专注在处理其他系统功能上。章节[3.9.1](#)描述了DMA的寄存器列表，章节[3.9.2](#)对DMA库函数进行说明。

3.9.1. 外设寄存器说明

DMA寄存器列表如下表所示：

表 3-165. DMA 寄存器

寄存器名称	寄存器描述
DMA_INTF	中断标志位寄存器
DMA_INTC	中断标志位清除器
DMA_CHxCTL (x=0..6)	通道x控制寄存器
DMA_CHxCNT (x=0..6)	通道x计数寄存器
DMA_CHxPADDR (x=0..6)	通道x外设基地址寄存器
DMA_CHxMADDR (x=0..6)	通道x存储器基地址寄存器

3.9.2. 外设库函数说明

DMA库函数列表如下表所示：

表 3-166. DMA 库函数

库函数名称	库函数描述
dma_deinit	复位外设DMAx的通道y的所有寄存器
dma_struct_para_init	将DMA结构体中所有参数初始化为默认值
dma_init	初始化外设DMAx的通道y
dma_circulation_enable	DMA循环模式使能
dma_circulation_disable	DMA循环模式禁能
dma_memory_to_memory_enable	存储器到存储器DMA传输使能
dma_memory_to_memory_disable	存储器到存储器DMA传输禁能
dma_channel_enable	外设DMAx的通道y传输使能
dma_channel_disable	外设DMAx的通道y传输禁能
dma_periph_address_config	DMAx通道y传输的外设基地址配置
dma_memory_address_config	DMAx通道y传输的存储器基地址配置
dma_transfer_number_config	配置DMAx通道y还有多少数据要传输
dma_transfer_number_get	获取DMAx通道y还有多少数据要传输

库函数名称	库函数描述
dma_priority_config	DMAx通道y的传输软件优先级配置
dma_memory_width_config	DMAx通道y传输的存储器数据宽度配置
dma_periph_width_config	DMAx通道y传输的外设数据宽度配置
dma_memory_increase_enable	DMAx通道y传输的存储器地址生成算法增量模式使能
dma_memory_increase_disable	DMAx通道y传输的存储器地址生成算法增量模式禁能
dma_periph_increase_enable	DMAx通道y传输的外设地址生成算法增量模式使能
dma_periph_increase_disable	DMAx通道y传输的外设地址生成算法增量模式禁能
dma_transfer_direction_config	DMAx通道y的传输方向配置
dma_flag_get	获取DMAx通道y标志位状态
dma_flag_clear	清除DMAx通道y标志位状态
dma_interrupt_flag_get	获取DMAx通道y中断标志位状态
dma_interrupt_flag_clear	清除DMAx通道y中断标志位状态
dma_interrupt_enable	DMAx通道y中断使能
dma_interrupt_disable	DMAx通道y中断禁能

结构体 **dma_parameter_struct**

表 3-167. 结构体 **dma_parameter_struct**

成员名称	功能描述
periph_addr	外设基地址
periph_width	外设数据传输宽度
memory_addr	存储器基地址
memory_width	存储器数据传输宽度
number	DMA通道数据传输数量
priority	DMA通道传输软件优先级
periph_inc	外设地址生成算法模式
memory_inc	存储器地址生成算法模式
direction	DMA通道数据传输方向

函数 **dma_deinit**

函数dma_deinit描述见下表:

表 3-168. 函数 **dma_deinit**

函数名称	dma_deinit
函数原型	void dma_deinit(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	复位外设DMAx的通道y的所有寄存器
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* DMA0 channel0 initialize */
dma_deinit(DMA0, DMA_CH0);
```

函数 **dma_struct_para_init**

函数dma_struct_para_init描述见下表:

表 3-169. 函数 **dma_struct_para_init**

函数名称	dma_struct_para_init
函数原型	void dma_struct_para_init(dma_parameter_struct* init_struct);
功能描述	将DMA结构体中所有参数初始化为默认值

先决条件	无
被调用函数	无
输入参数{in}	
*init_struct	一个已经定义的dma_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of DMA */
dma_parameter_struct dma_init_struct;
dma_struct_para_init(&dma_init_struct);
```

函数 **dma_init**

函数dma_init描述见下表：

表 3-170. 函数 **dma_init**

函数名称	dma_init
函数原型	void dma_init(uint32_t dma_periph, dma_channel_enum channelx, dma_parameter_struct* init_struct);
功能描述	初始化外设DMAx的通道y
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	

init_struct	初始化结构体，结构体成员参考 表 3-167. 结构体dma_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* DMA0 channel0 initialize */
dma_parameter_struct dma_init_struct;
dma_deinit(DMA0, DMA_CH0);
dma_struct_para_init(&dma_init_struct);

dma_init_struct.direction = DMA_PERIPHERAL_TO_MEMORY;
dma_init_struct.memory_addr = (uint32_t)g_destbuf;
dma_init_struct.memory_inc = DMA_MEMORY_INCREASE_ENABLE;
dma_init_struct.memory_width = DMA_MEMORY_WIDTH_8BIT;
dma_init_struct.number = TRANSFER_NUM;
dma_init_struct.periph_addr = (uint32_t)BANK0_WRITE_START_ADDR;
dma_init_struct.periph_inc = DMA_PERIPH_INCREASE_ENABLE;
dma_init_struct.periph_width = DMA_PERIPHERAL_WIDTH_8BIT;
dma_init_struct.priority = DMA_PRIORITY_ULTRA_HIGH;
dma_init(DMA0, DMA_CH0, &dma_init_struct);

```

函数 **dma_circulation_enable**

函数dma_circulation_enable描述见下表：

表 3-171. 函数 **dma_circulation_enable**

函数名称	dma_circulation_enable
函数原型	void dma_circulation_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA循环模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择

输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 circulation mode */
dma_circulation_enable(DMA0, DMA_CH0);
```

函数 **dma_circulation_disable**

函数dma_circulation_disable描述见下表：

表 3-172. 函数 **dma_circulation_disable**

函数名称	dma_circulation_disable
函数原型	void dma_circulation_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMA循环模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 circulation mode */
dma_circulation_disable(DMA0, DMA_CH0);
```

函数 **dma_memory_to_memory_enable**

函数dma_memory_to_memory_enable描述见下表：

表 3-173. 函数 **dma_memory_to_memory_enable**

函数名称	dma_memory_to_memory_enable
函数原型	void dma_memory_to_memory_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	存储器到存储器DMA传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 memory to memory mode */
dma_memory_to_memory_enable(DMA0, DMA_CH0);
```

函数 `dma_memory_to_memory_disable`

函数`dma_memory_to_memory_disable`描述见下表:

表 3-174. 函数 `dma_memory_to_memory_disable`

函数名称	<code>dma_memory_to_memory_disable</code>
函数原形	<code>void dma_memory_to_memory_disable(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	存储器到存储器DMA传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0, 1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</code>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable DMA0 channel0 memory to memory mode */  
dma_memory_to_memory_disable(DMA0, DMA_CH0);
```

函数 `dma_channel_enable`

函数`dma_channel_enable`描述见下表:

表 3-175. 函数 `dma_channel_enable`

函数名称	<code>dma_channel_enable</code>
函数原型	<code>void dma_channel_enable(uint32_t dma_periph, dma_channel_enum channelx);</code>

功能描述	外设DMAx的通道y传输使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA0 channel0 */
dma_channel_enable(DMA0, DMA_CH0);
```

函数 **dma_channel_disable**

函数dma_channel_disable描述见下表：

表 3-176. 函数 **dma_channel_disable**

函数名称	dma_channel_disable
函数原型	void dma_channel_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	外设DMAx的通道y传输禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设

<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable DMA0 channel0 */
dma_channel_disable(DMA0, DMA_CH0);
```

函数 **dma_periph_address_config**

函数dma_periph_address_config描述见下表：

表 3-177. 函数 **dma_periph_address_config**

函数名称	dma_periph_address_config
函数原型	void dma_periph_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx通道y传输的外设基地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择

输入参数{in}	
address	外设地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
#define BANK0_WRITE_START_ADDR ((uint32_t)0x08004000)

dma_periph_address_config(DMA0, DMA_CH0, BANK0_WRITE_START_ADDR);
```

函数 **dma_memory_address_config**

函数dma_memory_address_config描述见下表：

表 3-178. 函数 **dma_memory_address_config**

函数名称	dma_memory_address_config
函数原型	void dma_memory_address_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t address);
功能描述	DMAx通道y传输的存储器地址配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
address	存储器地址
输出参数{out}	

-	-
返回值	
-	-

例如：

```
uint8_t g_destbuf[TRANSFER_NUM];
dma_memory_address_config(DMA0, DMA_CH0, (uint32_t) g_destbuf);
```

函数 **dma_transfer_number_config**

函数dma_transfer_number_config描述见下表：

表 3-179. 函数 **dma_transfer_number_config**

函数名称	dma_transfer_number_config
函数原型	void dma_transfer_number_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t number);
功能描述	配置DMAx通道y还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
number	数据传输数量 (0x0 – 0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
#define TRANSFER_NUM          0x400
dma_transfer_number_config(DMA0, DMA_CH0, TRANSFER_NUM);
```

函数 **dma_transfer_number_get**

函数dma_transfer_number_get描述见下表：

表 3-180. 函数 **dma_transfer_number_get**

函数名称	dma_transfer_number_get
函数原型	uint32_t dma_transfer_number_get(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	获取DMA通道y还有多少数据要传输
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输出参数{out}	
-	-
返回值	
uint32_t	DMA数据传输剩余数量 (0x0 – 0xFFFF)

例如：

```
uint32_t number = 0;
number = dma_transfer_number_get(DMA0, DMA_CH0);
```

函数 **dma_priority_config**

函数dma_priority_config描述见下表：

表 3-181. 函数 `dma_priority_config`

函数名称	<code>dma_priority_config</code>
函数原型	<code>void dma_priority_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t priority);</code>
功能描述	DMAx通道y的传输软件优先级配置
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0,1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</code>	DMA通道选择
输入参数{in}	
<code>priority</code>	DMA通道软件优先级
<code>DMA_PRIORITY_LOW</code>	低优先级
<code>DMA_PRIORITY_MEDIUM</code>	中优先级
<code>DMA_PRIORITY_HIGH</code>	高优先级
<code>DMA_PRIORITY_ULTRA_HIGH</code>	极高优先级
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_priority_config(DMA0, DMA_CH0, DMA_PRIORITY_ULTRA_HIGH);
```

函数 `dma_memory_width_config`

函数`dma_memory_width_config`描述见下表：

表 3-182. 函数 `dma_memory_width_config`

函数名称	<code>dma_memory_width_config</code>
函数原型	<code>void dma_memory_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t mwidth);</code>
功能描述	DMAx通道y传输的存储器数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0,1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</code>	DMA通道选择
输入参数{in}	
<code>mwidth</code>	存储器数据传输宽度
<code>DMA_MEMORY_WIDTH_8BIT</code>	8位数据传输宽度
<code>DMA_MEMORY_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_MEMORY_WIDTH_32BIT</code>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_width_config(DMA0, DMA_CH0, DMA_MEMORY_WIDTH_8BIT);
```

函数 `dma_periph_width_config`

函数`dma_periph_width_config`描述见下表：

表 3-183. 函数 `dma_periph_width_config`

函数名称	<code>dma_periph_width_config</code>
函数原型	<code>void dma_periph_width_config (uint32_t dma_periph, dma_channel_enum channelx, uint32_t pwidth);</code>
功能描述	DMAx通道y传输的外设数据宽度配置
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0,1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</code>	DMA通道选择
输入参数{in}	
<code>pwidth</code>	外设数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_8BIT</code>	8位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_16BIT</code>	16位数据传输宽度
<code>DMA_PERIPHERAL_WIDTH_32BIT</code>	32位数据传输宽度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_periph_width_config(DMA0, DMA_CH0, DMA_PERIPHERAL_WIDTH_8BIT);
```

函数 `dma_memory_increase_enable`

函数`dma_memory_increase_enable`描述见下表:

表 3-184. 函数 `dma_memory_increase_enable`

函数名称	<code>dma_memory_increase_enable</code>
函数原型	<code>void dma_memory_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	DMAx通道y传输的存储器地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
<code>dma_periph</code>	DMA外设
<code>DMAx(x=0,1)</code>	DMA外设选择
输入参数{in}	
<code>channelx</code>	DMA通道
<code>DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)</code>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
dma_memory_increase_enable(DMA0, DMA_CH0);
```

函数 `dma_memory_increase_disable`

函数`dma_memory_increase_disable`描述见下表:

表 3-185. 函数 `dma_memory_increase_disable`

函数名称	<code>dma_memory_increase_disable</code>
函数原型	<code>void dma_memory_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);</code>
功能描述	DMAx通道y传输的存储器地址生成算法增量模式禁能

先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_memory_increase_disable(DMA0, DMA_CH0);
```

函数 **dma_periph_increase_enable**

函数**dma_periph_increase_enable**描述见下表：

表 3-186. 函数 **dma_periph_increase_enable**

函数名称	dma_periph_increase_enable
函数原型	void dma_periph_increase_enable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y传输的外设地址生成算法增量模式使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	

channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_periph_increase_enable(DMA0, DMA_CH0);
```

函数 **dma_periph_increase_disable**

函数dma_periph_increase_disable描述见下表：

表 3-187. 函数 **dma_periph_increase_disable**

函数名称	dma_periph_increase_disable
函数原型	void dma_periph_increase_disable(uint32_t dma_periph, dma_channel_enum channelx);
功能描述	DMAx通道y传输的外设地址生成算法增量模式禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
<i>DMAx(x=0,1)</i>	DMA外设选择
输入参数{in}	
channelx	DMA通道
<i>DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)</i>	DMA通道选择
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
dma_periph_increase_disable(DMA0, DMA_CH0);
```

函数 **dma_transfer_direction_config**

函数**dma_transfer_direction_config**描述见下表：

表 3-188. 函数 **dma_transfer_direction_config**

函数名称	dma_transfer_direction_config
函数原型	void dma_transfer_direction_config(uint32_t dma_periph, dma_channel_enum channelx, uint32_t direction);
功能描述	DMAx通道y的传输方向配置
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
direction	数据传输方向
DMA_PERIPHERAL_TO_MEMORY	读取外设中数据，写入存储器
DMA_MEMORY_TO_PERIPHERAL	读取存储器中数据，写入外设
输出参数{out}	
-	-
返回值	
-	-

例如：

```
dma_transfer_direction_config(DMA0, DMA_CH0, DMA_PERIPHERAL_TO_MEMORY);
```

函数 **dma_flag_get**

函数**dma_flag_get**描述见下表：

表 3-189. 函数 **dma_flag_get**

函数名称	dma_flag_get
函数原型	FlagStatus dma_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMAx通道y标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_FLAG_G	DMA通道全局中断标志
DMA_FLAG_FTF	DMA通道传输完成标志
DMA_FLAG_HTF	DMA通道半传输完成标志
DMA_FLAG_ERR	DMA通道错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
FlagStatus flag = RESET;
flag = dma_flag_get(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

函数 **dma_flag_clear**

函数dma_flag_clear描述见下表：

表 3-190. 函数 **dma_flag_clear**

函数名称	dma_flag_clear
函数原型	void dma_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMAx通道y标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_FLAG_G	DMA通道全局中断标志
DMA_FLAG_FTF	DMA通道传输完成标志
DMA_FLAG_HTF	DMA通道半传输完成标志
DMA_FLAG_ERR	DMA通道错误标志
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
dma_flag_clear(DMA0, DMA_CH0, DMA_FLAG_FTF);
```

函数 **dma_interrupt_flag_get**

函数dma_interrupt_flag_get描述见下表：

表 3-191. 函数 **dma_interrupt_flag_get**

函数名称	dma_interrupt_flag_get
函数原型	FlagStatus dma_interrupt_flag_get(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	获取DMAx通道y中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1:x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_FT	DMA通道传输完成中断标志
DMA_INT_FLAG_HT	DMA通道半传输完成中断标志
DMA_INT_FLAG_ER	DMA通道错误中断标志
输出参数{out}	
-	-

返回值	
FlagStatus	SET或RESET

例如：

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

函数 `dma_interrupt_flag_clear`

函数`dma_interrupt_flag_clear`描述见下表：

表 3-192. 函数 `dma_interrupt_flag_clear`

函数名称	dma_interrupt_flag_clear
函数原型	void dma_interrupt_flag_clear(uint32_t dma_periph, dma_channel_enum channelx, uint32_t flag);
功能描述	清除DMAx通道y中断标志位状态
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输入参数{in}	
flag	DMA标志
DMA_INT_FLAG_G	DMA通道全局中断标志
DMA_INT_FLAG_F TF	DMA通道传输完成中断标志
DMA_INT_FLAG_H TF	DMA通道半传输完成中断标志

DMA_INT_FLAG_E RR	DMA通道错误中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
if(dma_interrupt_flag_get(DMA0, DMA_CH3, DMA_INT_FLAG_FTF)){
    dma_interrupt_flag_clear(DMA0, DMA_CH3, DMA_INT_FLAG_G);
}
```

函数 **dma_interrupt_enable**

函数dma_interrupt_enable描述见下表：

表 3-193. 函数 **dma_interrupt_enable**

函数名称	dma_interrupt_enable
函数原型	void dma_interrupt_enable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx通道y中断使能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断

DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 interrupt configuration */

dma_interrupt_enable(DMA0, DMA_CH0, DMA_INT_FTF);
```

函数 **dma_interrupt_disable**

函数dma_interrupt_disable描述见下表：

表 3-194. 函数 **dma_interrupt_disable**

函数名称	dma_interrupt_disable
函数原型	void dma_interrupt_disable(uint32_t dma_periph, dma_channel_enum channelx, uint32_t source);
功能描述	DMAx通道y中断禁能
先决条件	无
被调用函数	无
输入参数{in}	
dma_periph	DMA外设
DMAx(x=0,1)	DMA外设选择
输入参数{in}	
channelx	DMA通道
DMA_CHx(DMA0:x =0..6; DMA1: x=0..4)	DMA通道选择
输入参数{in}	
source	DMA中断源
DMA_INT_FTF	DMA通道传输完成中断

DMA_INT_HTF	DMA通道半传输完成中断
DMA_INT_ERR	DMA通道错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* DMA0 channel0 interrupt configuration */
dma_interrupt_disable(DMA0, DMA_CH0, DMA_INT_FTF);
```

3.10. ENET

以太网模块包含10/100Mbps以太网MAC（媒体访问控制器），采用DMA优化数据帧的发送与接收性能，支持MII（媒体独立接口）与RMII（简化的媒体独立接口）两种与物理层(PHY)通讯的标准接口，实现以太网数据帧的发送与接收。章节[3.10.1](#)描述了ENET的寄存器列表，章节[3.10.2](#)对ENET库函数进行说明。

3.10.1. 外设寄存器描述

ENET寄存器列表如下表所示：

表 3-195. ENET 寄存器

寄存器名称	寄存器描述
ENET_MAC_CFG	MAC配置寄存器
ENET_MAC_FRMF	MAC帧过滤器寄存器
ENET_MAC_HLH	MAC hash列表高寄存器
ENET_MAC_HLL	MAC hash列表低寄存器
ENET_MAC_PHY_CTL	MAC PHY控制寄存器
ENET_MAC_PHY_DATA	MAC MII数据寄存器
ENET_MAC_FCTL	MAC流控寄存器
ENET_MAC_VLT	MAC VLAN标签寄存器

寄存器名称	寄存器描述
ENET_MAC_RWFF	MAC远程唤醒帧过滤器寄存器
ENET_MAC_WUM	MAC唤醒管理寄存器
ENET_MAC_DBG	MAC调试寄存器
ENET_MAC_INTF	MAC中断状态寄存器
ENET_MAC_INTMSK	MAC中断屏蔽寄存器
ENET_MAC_ADDR0H	MAC地址0高寄存器
ENET_MAC_ADDR0L	MAC地址0低寄存器
ENET_MAC_ADDR1H	MAC地址1高寄存器
ENET_MAC_ADDR1L	MAC地址1低寄存器
ENET_MAC_ADDT2H	MAC地址2高寄存器
ENET_MAC_ADDT2L	MAC地址2低寄存器
ENET_MAC_ADDR3H	MAC地址3高寄存器
ENET_MAC_ADDR3L	MAC地址3低寄存器
ENET_MAC_FCTH	MAC流控阈值寄存器
ENET_MSC_CTL	MSC控制寄存器
ENET_MSC_RINTF	MSC接收中断状态寄存器
ENET_MSC_TINTF	MSC发送中断状态寄存器
ENET_MSC_RINTMSK	MSC接收中断屏蔽寄存器
ENET_MSC_TINTMSK	MSC发送中断屏蔽寄存器
ENET_MSC_SCCN	MSC “1次冲突后发送”好”帧的计数器寄存器

寄存器名称	寄存器描述
T	
ENET_MSC_MSCKNT	MSC 1次以上冲突后发送“好”帧的计数器寄存器
ENET_MSC_TGFCNT	MSC发送“好”帧计数器寄存器
ENET_MSC_RFCCNT	MSC CRC错误接收帧计数器寄存器
ENET_MSC_RFAECNT	MSC对齐错误接收帧计数器寄存器
ENET_MSC_RGUFCONT	MSC “好”单播帧接收帧计数器寄存器
ENET_PTP_TSCTL	PTP时间戳控制寄存器
ENET_PTP_SSINC	PTP亚秒递增寄存器
ENET_PTP_TSH	PTP时间戳高寄存器
ENET_PTP_TSL	PTP时间戳低寄存器
ENET_PTP_TSUH	PTP时间戳高更新寄存器
ENET_PTP_TSUL	PTP时间戳低更新寄存器
ENET_PTP_TSADDEND	PTP时间戳加数寄存器
ENET_PTP_ETH	PTP期望时间高寄存器
ENET_PTP_ETL	PTP期望时间低寄存器
ENET_PTP_TSFLAG	PTP时间戳标志寄存器
ENET_PTP_PPSCTL	PTP PPS控制寄存器
ENET_DMA_BCTL	DMA总线控制寄存器
ENET_DMA_TPEN	DMA发送查询使能寄存器
ENET_DMA_RPEN	DMA接收查询使能寄存器
ENET_DMA_RDTADDR	DMA接收描述符列表地址寄存器
ENET_DMA_TDTADDR	DMA发送描述符列表地址寄存器

寄存器名称	寄存器描述
ENET_DMA_STAT	DMA状态寄存器
ENET_DMA_CTL	DMA控制寄存器
ENET_DMA_INTEN	DMA中断使能寄存器
ENET_DMA_MFBO CNT	DMA丢失帧和缓存溢出计数器寄存器
ENET_DMA_RSWD C	DMA接收接收状态看门狗计数器寄存器
ENET_DMA_CTDA DDR	DMA当前发送描述符地址寄存器
ENET_DMA_CRDA DDR	DMA当前接收描述符地址寄存器
ENET_DMA_CTBA DDR	DMA当前发送缓存地址寄存器
ENET_DMA_CRBA DDR	DMA当前接收缓存地址寄存器

3.10.2. 外设库函数说明

ENET库函数列表如下表所示：

表 3-196. ENET 库函数

库函数名称	库函数描述
常用函数	
enet_deinit	复位ENET模块及相关软件初始化所需结构体
enet_initpara_config	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
enet_init	ENET模块初始化，配置用户最关心的功能
enet_software_reset	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
enet_rxframe_size_get	检测接收帧是否有错误，正确时返回帧长度
enet_descriptors_chain_init	初始化DMA接收/发送描述符为链模式
enet_descriptors_ring_init	初始化DMA接收/发送描述符为环模式
enet_frame_receive	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷

库函数名称	库函数描述
	贝到指定区域
enet_frame_transmit	将指定区域内的数据拷贝到当前发送描述符中，并发送
enet_transmit_checksum_config	配置发送帧校验和模式
enet_enable	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
enet_disable	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
enet_mac_address_set	配置MAC地址
enet_mac_address_get	获取MAC地址
enet_flag_get	获取ENET模块MAC/MSC/PTP/DMA状态标志位
enet_flag_clear	清除ENET状态标志位
enet_interrupt_enable	使能ENET模块MAC/MSC/DMA中断
enet_interrupt_disable	禁能ENET模块MAC/MSC/DMA中断
enet_interrupt_flag_get	获取ENET模块MAC/MSC/DMA中断标志位
enet_interrupt_flag_clear	禁能ENET模块DMA中断标志位
MAC功能函数	
enet_tx_enable	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
enet_tx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_rx_enable	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
enet_rx_disable	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
enet_registers_get	获取指定范围ENET寄存器值
enet_debug_status_get	获取ENET调试状态信息
enet_address_filter_enable	MAC地址过滤器使能
enet_address_filter_disable	MAC地址过滤器禁能
enet_address_filter_config	配置MAC地址过滤器模式
enet_phy_config	PHY接口配置（配置SMI时钟并复位PHY芯片）
enet_phy_write_read	写/读PHY寄存器
enet_phyloopback_enable	使能PHY芯片回环模式

库函数名称	库函数描述
enet_phyloopback_disable	禁能PHY芯片回环模式
enet_forward_feature_enable	使能ENET帧通过相关功能
enet_forward_feature_disable	禁能ENET帧通过相关功能
enet_filter_feature_enable	使能ENET帧过滤器相关功能
enet_filter_feature_disable	禁能ENET帧过滤器相关功能
流控功能函数	
enet_pauseframe_generate	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
enet_pauseframe_detect_config	配置暂停帧检测类型
enet_pauseframe_config	配置暂停帧参数
enet_flowcontrol_threshold_config	配置流控阈值
enet_flowcontrol_feature_enable	使能ENET流控相关功能
enet_flowcontrol_feature_disable	禁能ENET流控相关功能
DMA功能函数	
enet_dmaprocess_state_get	获取DMA发送/接收流程状态
enet_dmaprocess_resume	DMA发送/接收查询使能
enet_rxprocess_check_recovery	检测并恢复接收流程
enet_txfifo_flush	刷新ENET发送FIFO，并等待刷新操作完成
enet_current_desc_address_get	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
enet_desc_information_get	获取发送/接收描述符详细信息
enet_missed_frame_counter_get	获取接收丢弃帧数
描述符功能函数	
enet_desc_flag_get	获取ENET模块DMA描述符标志位
enet_desc_flag_set	设置ENET模块DMA描述符标志位
enet_desc_flag_clear	清除ENET模块DMA描述符标志位
enet_rx_desc_immediate_receive_complete_interrupt	接收完成后立即置位ENET_DMA_STAT寄存器的RS位
enet_rx_desc_delay_receive_complet	接收完成后延迟指定时间再置位ENET_DMA_STAT寄存器的

库函数名称	库函数描述
e_interrupt	RS位
enet_rxframe_drop	丢弃当前接收到的帧
enet_dma_feature_enable	使能ENET模块DMA相关功能
enet_dma_feature_disable	禁能ENET模块DMA相关功能
增强型描述符功能函数	
enet_rx_desc_enhanced_status_get	获取接收描述符增强状态标志位信息
enet_desc_select_enhanced_mode	配置DMA描述符为增强型描述符
enet_ptp_enhanced_descriptors_chain_init	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
enet_ptp_enhanced_descriptors_ring_init	初始化具有PTP功能的增强型DMA接收/发送描述符为环模式
enet_ptpframe_receive_enhanced_mode	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_enhanced_mode	在PTP模式下将指定区域内的数据拷贝到当前增强型发送描述符中，并同时间戳一起发送
常规型描述符功能函数	
enet_desc_select_normal_mode	配置DMA描述符为常规型描述符
enet_ptp_normal_descriptors_chain_init	初始化具有PTP功能的普通型DMA接收/发送描述符为链模式
enet_ptp_normal_descriptors_ring_init	初始化具有PTP功能的普通型DMA接收/发送描述符为环模式
enet_ptpframe_receive_normal_mode	在PTP模式下处理当前接收到的帧，并将当前普通型描述符中存储的接收帧数据和时间戳拷贝到指定区域
enet_ptpframe_transmit_normal_mode	在PTP模式下将指定区域内的数据拷贝到当前普通型发送描述符中，并同时间戳一起发送
WUM功能函数	
enet_wum_filter_register_pointer_reset	远程唤醒帧过滤器寄存器指针复位
enet_wum_filter_config	配置远程唤醒帧寄存器
enet_wum_feature_enable	使能ENET模块唤醒管理相关功能
enet_wum_feature_disable	禁能ENET模块唤醒管理相关功能

库函数名称	库函数描述
MSC功能函数	
enet_msc_counters_reset	复位MAC统计计数器组
enet_msc_feature_enable	使能MAC统计计数器相关功能
enet_msc_feature_disable	禁能MAC统计计数器相关功能
enet_msc_counters_preset_config	配置MAC统计计数器的预设模式
enet_msc_counters_get	获取MAC相关统计计数器值
PTP功能函数	
enet_ptp_subsecond_2_nanosecond	将亚秒值变为纳秒值
enet_ptp_nanosecond_2_subsecond	将纳秒值变为亚秒值
enet_ptp_feature_enable	使能PTP相关功能
enet_ptp_feature_disable	禁能PTP相关功能
enet_ptp_timestamp_function_config	配置PTP时间戳相关功能
enet_ptp_subsecond_increment_config	配置PTP系统时间亚秒增加值
enet_ptp_timestamp_addend_config	精调模式下PTP时钟频率校准配置
enet_ptp_timestamp_update_config	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
enet_ptp_expected_time_config	配置PTP期望时间
enet_ptp_system_time_get	获取PTP当前系统时间
enet_ptp_pps_output_frequency_config	配置PPS输出频率
enet_ptp_start	配置和启动PTP时间戳计数器
enet_ptp_finecorrection_adfreq	通过PTP时间戳加数寄存器精调系统时间
enet_ptp_coarsecorrection_systime_update	粗调系统时间
enet_ptp_finecorrection_settime	精调系统时间
enet_ptp_flag_get	获取PTP标志位状态
内部函数	
enet_initpara_reset	复位 ENET initpara struct, 需在enet_initpara_config()函数前

库函数名称	库函数描述
	调用

结构体 enet_initpara_struct

表 3-197. 结构体 enet_initpara_struct

成员名称	功能描述
option_enable	选择配置功能
forward_frame	发送帧相关配置参数
dmabus_mode	DMA总线模式相关配置参数
dma_maxburst	DMA最大传输相关配置参数
dma_arbitration	DMA接收/发送仲裁相关配置参数
store_forward_mode	存储转发模式相关配置参数
dma_function	DMA控制相关配置参数
vlan_config	VLAN标签相关配置参数
flow_control	流控相关配置参数
hashtable_high	hash列表高32位相关配置参数
hashtable_low	hash列表低32位相关配置参数
framesfilter_mode	帧过滤器控制相关配置参数
halfduplex_param	半双工相关配置参数
timer_config	帧发送计数器相关配置参数
interframegap	帧间隔相关配置参数

结构体 enet_descriptors_struct

表 3-198. 结构体 enet_descriptors_struct

成员名称	功能描述
status	描述符状态位
control_buffer_size	描述符控制位及缓冲区1、2长度
buffer1_addr	缓冲区1地址指针/时间戳低
buffer2_next_desc_addr	缓冲区2或下一描述符地址指针/时间戳高

extended_status	增强型描述符状态
reserved	保留
timestamp_low	增强型描述符时间戳低位
timestamp_high	增强型描述符时间戳高位

结构体 **enet_ptp_systime_struct**

表 3-199. 结构体 **enet_ptp_systime_struct**

成员名称	功能描述
second	系统时间（单位秒）
subsecond	系统时间（单位纳秒）
sign	系统时间符号位

函数 **enet_deinit**

函数 **enet_deinit** 描述见下表：

表 3-200. 函数 **enet_deinit**

函数名称	enet_deinit
函数原型	void enet_deinit(void);
功能描述	复位ENET模块及相关软件初始化所需结构体
先决条件	-
被调用函数	rcu_periph_reset_enable()/rcu_periph_reset_disable()/enet_initpara_reset()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the ENET */
enet_deinit();
```

函数 enet_initpara_config

函数enet_initpara_config描述见下表：

表 3-201. 函数 enet_initpara_config

函数名称	enet_initpara_config
函数原型	void enet_initpara_config(enet_option_enum option, uint32_t para)
功能描述	配置ENET模块的各类不常用功能。当enet_init()函数无法满足所需实现功能时调用，必须在enet_init()函数之前调用
先决条件	enet_initpara_reset(void)
被调用函数	-
输入参数{in}	
option	ENET模块功能选项，根据选择需要使用不同参数进行配置，下列参数仅可选择一个
<i>FORWARD_OPTION</i>	选择配置帧通过功能相关参数
<i>DMABUS_OPTION</i>	选择配置DMA总线模式相关参数
<i>DMA_MAXBURST_OPTION</i>	选择配置DMA最大突发传输相关参数
<i>DMA_ARBITRATION_OPTION</i>	选择配置DMA仲裁相关参数
<i>STORE_OPTION</i>	选择配置存储转发模式相关参数
<i>DMA_OPTION</i>	选择配置DMA相关参数
<i>VLAN_OPTION</i>	选择配置VLAN相关参数
<i>FLOWCTL_OPTION</i>	选择配置流控相关参数
<i>HASHH_OPTION</i>	选择配置HASH_H相关参数
<i>HASHL_OPTION</i>	选择配置HASH_L相关参数
<i>FILTER_OPTION</i>	选择配置帧过滤器相关参数
<i>HALFDUPLEX_OPTION</i>	选择配置半双工模式相关参数
<i>TIMER_OPTION</i>	选择配置计数器相关参数
<i>INTERFRAMEGAP_OPTION</i>	选择配置帧间隔相关参数

输入参数{in}	
para (该参数值需根据 option 参数对应值进行选取)	下列参数值均可以被配置 例如: para = (value1 value2 value3...)
当 option 参数值为 FORWARD_OPTION 时	
value1	<i>ENET_AUTO_PADCRC_DROP_ENABLE / ENET_AUTO_PADCRC_DROP_DISABLE</i>
value2	<i>ENET_TYPEFRAME_CRC_DROP_ENABLE / ENET_TYPEFRAME_CRC_DROP_DISABLE</i>
value3	<i>ENET_FORWARD_ERRFRAMES_ENABLE / ENET_FORWARD_ERRFRAMES_DISABLE</i>
value4	<i>ENET_FORWARD_UNDERSZ_GOODFRAMES_ENABLE / ENET_FORWARD_UNDERSZ_GOODFRAMES_DISABLE</i>
当 option 参数值为 DMABUS_OPTION 时	
value1	<i>ENET_ADDRESS_ALIGN_ENABLE / ENET_ADDRESS_ALIGN_DISABLE</i>
value2	<i>ENET_FIXED_BURST_ENABLE / ENET_FIXED_BURST_DISABLE</i>
value3	<i>ENET_MIXED_BURST_ENABLE / ENET_MIXED_BURST_DISABLE</i>
当 option 参数值为 DMA_MAXBURST_OPTION 时	
value1	<i>ENET_RXDP_1BEAT / ENET_RXDP_2BEAT / ENET_RXDP_4BEAT / ENET_RXDP_8BEAT / ENET_RXDP_16BEAT / ENET_RXDP_32BEAT / ENET_RXDP_4xPGBL_4BEAT / ENET_RXDP_4xPGBL_8BEAT / ENET_RXDP_4xPGBL_16BEAT / ENET_RXDP_4xPGBL_32BEAT / ENET_RXDP_4xPGBL_64BEAT / ENET_RXDP_4xPGBL_128BEAT</i>
value2	<i>ENET_PGBL_1BEAT / ENET_PGBL_2BEAT / ENET_PGBL_4BEAT / ENET_PGBL_8BEAT / ENET_PGBL_16BEAT / ENET_PGBL_32BEAT / ENET_PGBL_4xPGBL_4BEAT / ENET_PGBL_4xPGBL_8BEAT / ENET_PGBL_4xPGBL_16BEAT / ENET_PGBL_4xPGBL_32BEAT / ENET_PGBL_4xPGBL_64BEAT / ENET_PGBL_4xPGBL_128BEAT</i>
value3	<i>ENET_RXTX_DIFFERENT_PGBL / ENET_RXTX_SAME_PGBL</i>
当 option 参数值为 DMA_ARBITRATION_OPTION 时	
value1	<i>ENET_ARBITRATION_RXPRIORTX</i>
value2	<i>ENET_ARBITRATION_RXTX_1_1 / ENET_ARBITRATION_RXTX_2_1 / ENET_ARBITRATION_RXTX_3_1 / ENET_ARBITRATION_RXTX_4_1</i>

当 option 参数值为 STORE_OPTION 时	
value1	<i>ENET_RX_MODE_STOREFORWARD / ENET_RX_MODE_CUTTHROUGH</i>
value2	<i>ENET_TX_MODE_STOREFORWARD / ENET_TX_MODE_CUTTHROUGH</i>
value3	<i>ENET_RX_THRESHOLD_64BYTES / ENET_RX_THRESHOLD_32BYTES / ENET_RX_THRESHOLD_96BYTES / ENET_RX_THRESHOLD_128BYTES</i>
value4	<i>ENET_TX_THRESHOLD_64BYTES / ENET_TX_THRESHOLD_128BYTES / ENET_TX_THRESHOLD_192BYTES / ENET_TX_THRESHOLD_256BYTES / ENET_TX_THRESHOLD_40BYTES / ENET_TX_THRESHOLD_32BYTES / ENET_TX_THRESHOLD_24BYTES / ENET_TX_THRESHOLD_16BYTES</i>
当 option 参数值为 DMA_OPTION 时	
value1	<i>ENET_FLUSH_RXFRAME_ENABLE / ENET_FLUSH_RXFRAME_DISABLE</i>
value2	<i>ENET_SECONDFRAME_OPT_ENABLE / ENET_SECONDFRAME_OPT_DISABLE</i>
value3	<i>ENET_ENHANCED_DESCRIPTOR / ENET_NORMAL_DESCRIPTOR</i>
当 option 参数值为 VLAN_OPTION 时	
value1	<i>ENET_VLANTAGCOMPARISON_12BIT / ENET_VLANTAGCOMPARISON_16BIT</i>
value2	<i>MAC_VLT_VLTI(regval)</i>
当 option 参数值为 FLOWCTL_OPTION 时	
value1	<i>MAC_FCTL_PTM(regval)</i>
value2	<i>ENET_ZERO_QUANTA_PAUSE_ENABLE / ENET_ZERO_QUANTA_PAUSE_DISABLE</i>
value3	<i>ENET_PAUSETIME_MINUS4 / ENET_PAUSETIME_MINUS28 / ENET_PAUSETIME_MINUS144 / ENET_PAUSETIME_MINUS256</i>
value4	<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT / ENET_UNIQUE_PAUSEDTECT</i>
value5	<i>ENET_RX_FLOWCONTROL_ENABLE / ENET_RX_FLOWCONTROL_DISABLE</i>
value6	<i>ENET_TX_FLOWCONTROL_ENABLE / ENET_TX_FLOWCONTROL_DISABLE</i>
当 option 参数值为 HASHH_OPTION 时	
value1	<i>0x0~0xFFFF FFFFU</i>

当 option 参数值为 <i>HASHL_OPTION</i> 时	
value1	0x0~0xFFFF FFFFU
当 option 参数值为 <i>FILTER_OPTION</i> 时	
value1	<i>ENET_SRC_FILTER_NORMAL_ENABLE</i> / <i>ENET_SRC_FILTER_INVERSE_ENABLE</i> / <i>ENET_SRC_FILTER_DISABLE</i>
value2	<i>ENET_DEST_FILTER_INVERSE_ENABLE</i> / <i>ENET_DEST_FILTER_INVERSE_DISABLE</i>
value3	<i>ENET_MULTICAST_FILTER_HASH_OR_PERFECT</i> / <i>ENET_MULTICAST_FILTER_HASH</i> / <i>ENET_MULTICAST_FILTER_PERFECT</i> / <i>ENET_MULTICAST_FILTER_NONE</i>
value4	<i>ENET_UNICAST_FILTER_EITHER</i> / <i>ENET_UNICAST_FILTER_HASH</i> / <i>ENET_UNICAST_FILTER_PERFECT</i>
value5	<i>ENET_PCFRM_PREVENT_ALL</i> / <i>ENET_PCFRM_PREVENT_PAUSEFRAME</i> / <i>ENET_PCFRM_FORWARD_ALL</i> / <i>ENET_PCFRM_FORWARD_FILTERED</i>
当 option 参数值为 <i>HALFDUPLEX_OPTION</i> 时	
value1	<i>ENET_CARRIERSENSE_ENABLE</i> / <i>ENET_CARRIERSENSE_DISABLE</i>
value2	<i>ENET_RECEIVEOWN_ENABLE</i> / <i>ENET_RECEIVEOWN_DISABLE</i>
value3	<i>ENET_RETRYTRANSMISSION_ENABLE</i> / <i>ENET_RETRYTRANSMISSION_DISABLE</i>
value4	<i>ENET_BACKOFFLIMIT_10</i> / <i>ENET_BACKOFFLIMIT_8</i> / <i>ENET_BACKOFFLIMIT_4</i> / <i>ENET_BACKOFFLIMIT_1</i>
value5	<i>ENET_DEFERRALCHECK_ENABLE</i> / <i>ENET_DEFERRALCHECK_DISABLE</i>
当 option 参数值为 <i>TIMER_OPTION</i> 时	
value1	<i>ENET_WATCHDOG_ENABLE</i> / <i>ENET_WATCHDOG_DISABLE</i>
value2	<i>ENET_JABBER_ENABLE</i> / <i>ENET_JABBER_DISABLE</i>
当 option 参数值为 <i>INTERFRAMEGAP_OPTION</i> 时	
value1	<i>ENET_INTERFRAMEGAP_96BIT</i> / <i>ENET_INTERFRAMEGAP_88BIT</i> / <i>ENET_INTERFRAMEGAP_80BIT</i> / <i>ENET_INTERFRAMEGAP_72BIT</i> / <i>ENET_INTERFRAMEGAP_64BIT</i> / <i>ENET_INTERFRAMEGAP_56BIT</i> / <i>ENET_INTERFRAMEGAP_48BIT</i> / <i>ENET_INTERFRAMEGAP_40BIT</i>
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* config DMA option of the ENET */
enet_initpara_reset();

enet_initpara_config(DMA_OPTION, ENET_FLUSH_RXFRAME_ENABLE | ENET_SECONDFRAME_OPT_ENABLE | ENET_NORMAL_DESCRIPTOR);
```

函数 enet_init

函数enet_init描述见下表：

表 3-202. 函数 enet_init

函数名称	enet_init
函数原型	ErrStatus enet_init(enet_mediemode_enum mediemode, enetc_checksumconf_enum checksum, enet_frmrecept_enum recept);
功能描述	ENET模块初始化，配置用户最关心的功能
先决条件	enet_deinit()
被调用函数	enet_phy_config()/enet_phy_write_read()/enet_default_init()
输入参数{in}	
mediemode	ENET通讯方式配置，仅可选择唯一参数
ENET_AUTO_NEGOTIATION	PHY自协商
ENET_100M_FULLDUPLEX	100Mbit/s, 全双工
ENET_100M_HALFDUPLEX	100Mbit/s, 半双工
ENET_10M_FULLDUPLEX	10Mbit/s, 全双工
ENET_10M_HALFDUPLEX	10Mbit/s, 半双工
ENET_LOOPBACK_MODE	MII模式下的回环模式

输入参数{in}	
checksum	IP帧数据校验和功能，仅可选择唯一参数
<i>ENET_NO_AUTOCH_ECKSUM</i>	关闭IP帧校验和功能
<i>ENET_AUTOCHECKSUM_DROP_FAILFRAMES</i>	使能IP帧校验和功能
<i>ENET_AUTOCHECKSUM_ACCEPT_FAILFRAMES</i>	使能IP帧校验和功能，不丢弃仅有载荷错误的帧
输入参数{in}	
recept	帧过滤功能，仅可选择唯一参数
<i>ENET_PROMISCUOUS_MODE</i>	使能混杂模式
<i>ENET_RECEIVEALL</i>	接收所有帧
<i>ENET_BROADCAST_FRAMES_PASS</i>	接收广播帧
<i>ENET_BROADCAST_FRAMES_DROP</i>	禁止接收广播帧
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```

/* initialize ENET peripheral */

ErrStatus enet_init_status;

enet_init_status = enet_init(ENET_AUTO_NEGOTIATION, ENET_AUTOCHECKSUM_DROP_FAILFRAMES, ENET_BROADCAST_FRAMES_PASS);

```

函数 **enet_software_reset**

函数**enet_software_reset**描述见下表：

表 3-203. 函数 enet_software_reset

函数名称	enet_software_reset
函数原型	ErrStatus enet_software_reset(void);
功能描述	复位ENET寄存器，并检测CLK_TX/CLK_RX信号
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* reset all core internal registers located in CLK_TX and CLK_RX */

ErrStatus reval_state = ERROR;

reval_state = enet_software_reset();
```

函数 enet_rxframe_size_get

函数enet_rxframe_size_get描述见下表：

表 3-204. 函数 enet_rxframe_size_get

函数名称	enet_rxframe_size_get
函数原型	uint32_t enet_rxframe_size_get(void);
功能描述	检测接收帧是否有错误，正确时返回帧长度
先决条件	-
被调用函数	enet_rxframe_drop()
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
uint32_t	取值范围: 0x0 - 0x3FFF

例如:

```
/* check receive frame valid */

uint32_t reval;

reval = enet_rxframe_size_get();
```

函数 **enet_descriptors_chain_init**

函数enetc_descriptors_chain_init描述见下表:

表 3-205. 函数 **enet_descriptors_chain_init**

函数名称	enet_descriptors_chain_init
函数原型	void enet_descriptors_chain_init(enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	想要初始化的描述符类型, 下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the DMA Tx/Rx descriptors's parameters in chain mode */

enet_descriptors_chain_init(ENET_DMA_TX);
```

函数 **enet_descriptors_ring_init**

函数enetc_descriptors_ring_init描述见下表:

表 3-206. 函数 enet_descriptors_ring_init

函数名称	enet_descriptors_ring_init
函数原型	void enet_descriptors_ring_init(enet_dmadirection_enum direction);
功能描述	初始化DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	想要初始化的描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx/Rx descriptors's parameters in ring mode */
enet_descriptors_ring_init(ENET_DMA_TX);
```

函数 enet_frame_receive

函数enet_frame_receive描述见下表：

表 3-207. 函数 enet_frame_receive

函数名称	enet_frame_receive
函数原型	ErrStatus enet_frame_receive(uint8_t *buffer, uint32_t bufsize);
功能描述	处理当前接收到的帧，并将当前描述符中存储的接收帧数据拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区长度，范围(0~1524)
输出参数{out}	

buffer	接收帧数据的缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将数据拷贝到用户缓冲区内
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* transfer received frame data to application buffer */

uint8_t data_buffer[1500];

uint32_t data_size;

enet_frame_receive(data_buffer, &data_size);
```

函数 enet_frame_transmit

函数enet_frame_transmit描述见下表：

表 3-208. 函数 enet_frame_transmit

函数名称	enet_frame_transmit
函数原型	ErrStatus enet_frame_transmit(uint8_t *buffer, uint32_t length);
功能描述	将指定区域内的数据拷贝到当前发送描述符中，并发送
先决条件	-
被调用函数	-
输入参数{in}	
buffer	等待发送的帧数据缓冲区地址指针。如果输入NULL，用户需要在调用该函数之前将待发送数据拷贝到描述符指定的位置
输入参数{in}	
length	待发送数据长度，范围(0~1524)
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* transfer buffer data of application */

uint8_t data_buffer[1500];
```

```

uint32_t data_size = 800;
enet_frame_transmit (data_buffer, data_size);

```

函数 enet_transmit_checksum_config

函数enet_transmit_checksum_config描述见下表:

表 3-209. 函数 enet_transmit_checksum_config

函数名称	enet_transmit_checksum_config
函数原型	void enet_transmit_checksum_config(enet_descriptors_struct *desc, uint32_t checksum);
功能描述	配置发送帧校验和模式
先决条件	-
被调用函数	-
输入参数{in}	
desc	需要配置的描述符地址指针, 结构体成员介绍参考 结构体 enet descriptors struct
输入参数{in}	
checksum	IP帧校验和配置, 下列参数仅可选择一个
ENET_CHECKSUM_DISABLE	禁能校验和自动插入
ENET_CHECKSUM_IPV4HEADER	仅使能IP头校验和计算和插入
ENET_CHECKSUM_TCUPDPICMP_SEGMENT	TCP/UDP/ICMP校验和(除去伪报头)计算和插入
ENET_CHECKSUM_TCUPDPICMP_FULL	TCP/UDP/ICMP校验和计算和插入
输出参数{out}	
返回值	

例如:

```

/* configure the transmit IP frame checksum offload calculation and insertion */

enet_descriptors_struct rx_desc;

enet_transmit_checksum_config(rx_desc, ENET_CHECKSUM_TCPUDPICMP_FULL);

```

函数 enet_enable

函数enet_enable描述见下表:

表 3-210. 函数 enet_enable

函数名称	enet_enable
函数原型	void enet_enable(void);
功能描述	ENET Tx/Rx功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_tx_enable()/enet_rx_enable()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable the ENET */

enet_enable();

```

函数 enet_disable

函数enet_disable描述见下表:

表 3-211. 函数 enet_disable

函数名称	enet_disable
函数原型	void enet_disable(void);
功能描述	ENET Tx/Rx功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the ENET */
enet_disable();
```

函数 **enet_mac_address_set**

函数 **enet_mac_address_set** 描述见下表：

表 3-212. 函数 **enet_mac_address_set**

函数名称	enet_mac_address_set
函数原型	void enet_mac_address_set(enet_macaddress_enum mac_addr, uint8_t paddr[]);
功能描述	配置MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择何组MAC地址将被配置，下列参数仅可选择一个
ENET_MAC_ADDRESS0	配置MAC address 0过滤器
ENET_MAC_ADDRESS1	配置MAC address 1过滤器
ENET_MAC_ADDRESS2	配置MAC address 2过滤器
ENET_MAC_ADDRESS3	配置MAC address 3过滤器
输入参数{in}	
paddr	存储MAC地址的缓冲区指针，小端存储

	例如MAC地址为aa:bb:cc:dd:ee:22, 缓冲区内数据为{22, ee, dd, cc, bb, aa}
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config mac address */

netif->hwaddr[0] = 0x02;

netif->hwaddr[1] = 0xaa;

netif->hwaddr[2] = 0xbb;

netif->hwaddr[3] = 0xcc;

netif->hwaddr[4] = 0xdd;

netif->hwaddr[5] = 0xee;

enet_mac_address_set(ENET_MAC_ADDRESS0, netif->hwaddr);
```

函数 enet_mac_address_get

函数enet_mac_address_get描述见下表：

表 3-213. 函数 enet_mac_address_get

函数名称	enet_mac_address_get
函数原型	void enet_mac_address_get(enet_macaddress_enum mac_addr, uint8_t paddr[])
功能描述	获取MAC地址
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择何组MAC地址将被配置, 下列参数仅可选择一个
ENET_MAC_ADDRESS0	配置MAC address 0过滤器
ENET_MAC_ADDRESS1	配置MAC address 1过滤器

<i>ENET_MAC_ADDRESS2</i>	配置MAC address 2过滤器
<i>ENET_MAC_ADDRESS3</i>	配置MAC address 3过滤器
输出参数{out}	
<i>paddr</i>	存储MAC地址的缓冲区指针，小端存储 例如MAC地址为aa:bb:cc:dd:ee:22，缓冲区内数据为{22, ee, dd, cc, bb, aa}
返回值	
-	-

例如：

```
/* get mac address */
enet_mac_address_get (ENET_MAC_ADDRESS0, netif->hwaddr);
```

函数 **enet_flag_get**

函数enet_flag_get描述见下表：

表 3-214. 函数 **enet_flag_get**

函数名称	enet_flag_get
函数原型	FlagStatus enet_flag_get(enet_flag_enum enet_flag);
功能描述	获取ENET模块MAC/MSC/PTP/DMA状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
<i>enet_flag</i>	ENET状态标志位，下列参数仅可选择一个
<i>ENET_MAC_FLAG_MPKR</i>	接收到魔术帧标志位
<i>ENET_MAC_FLAG_WUFR</i>	接收到唤醒帧标志位
<i>ENET_MAC_FLAG_FLOWCONTROL</i>	流控状态标志位
<i>ENET_MAC_FLAG_WUM</i>	WUM状态标志位

<i>ENET_MAC_FLAG_MSC</i>	MSC状态标志位
<i>ENET_MAC_FLAG_MSCR</i>	MSC接收状态标志位
<i>ENET_MAC_FLAG_MSCT</i>	MSC发送状态标志位
<i>ENET_MAC_FLAG_TMST</i>	时间戳触发状态标志位
<i>ENET_PTP_FLAG_TSSCO</i>	时间戳秒计数溢出标志位
<i>ENET_PTP_FLAG_TTM</i>	目标时间匹配标志位
<i>ENET_MSC_FLAG_RFCE</i>	接收帧CRC错误标志位
<i>ENET_MSC_FLAG_RFAE</i>	接收帧对齐错误标志位
<i>ENET_MSC_FLAG_RGUF</i>	接收到“好”的单播帧标志位
<i>ENET_MSC_FLAG_TGFSC</i>	发送“好”的帧时仅遇到1个冲突标志位
<i>ENET_MSC_FLAG_TGFMSC</i>	发送“好”的帧时遇到1个以上冲突
<i>ENET_MSC_FLAG_TGF</i>	发送“好”的帧标志位
<i>ENET_DMA_FLAG_TS</i>	发送状态标志位
<i>ENET_DMA_FLAG_TPS</i>	发送流程停止状态标志位
<i>ENET_DMA_FLAG_TBU</i>	发送缓冲区不可用状态标志位
<i>ENET_DMA_FLAG_TJT</i>	发送jabber超时状态标志位
<i>ENET_DMA_FLAG_RO</i>	接收溢出状态标志位

<i>ENET_DMA_FLAG_TU</i>	发送下溢状态标志位
<i>ENET_DMA_FLAG_RS</i>	接收状态标志位
<i>ENET_DMA_FLAG_RBU</i>	接收缓冲区不可用状态标志位
<i>ENET_DMA_FLAG_RPS</i>	接受流程停止状态标志位
<i>ENET_DMA_FLAG_RWT</i>	接收看门狗超时状态标志位
<i>ENET_DMA_FLAG_ET</i>	早发送状态标志位
<i>ENET_DMA_FLAG_FBE</i>	致命总线错误状态标志位
<i>ENET_DMA_FLAG_ER</i>	早接收状态标志位
<i>ENET_DMA_FLAG_AI</i>	异常中断汇总标志位
<i>ENET_DMA_FLAG_NI</i>	正常中断汇总标志位
<i>ENET_DMA_FLAG_EB_DMA_ERROR</i>	DMA错误标志位
<i>ENET_DMA_FLAG_EB_TRANSFER_ER_ROR</i>	发送错误标志位
<i>ENET_DMA_FLAG_EB_ACCESS_ERRO_R</i>	DMA访问错误标志位
<i>ENET_DMA_FLAG_MSC</i>	MSC状态标志位
<i>ENET_DMA_FLAG_WUM</i>	WUM状态标志位
<i>ENET_DMA_FLAG_TST</i>	时间戳触发状态标志位
输出参数{out}	

-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* check whether the specified flag bit is set */

enet_flag_get (ENET_DMA_FLAG_RS);
```

函数 enet_flag_clear

函数enet_flag_clear描述见下表：

表 3-215. 函数 enet_flag_clear

函数名称	enet_flag_clear
函数原型	void enet_flag_clear(enet_flag_clear_enum enet_flag);
功能描述	清除ENET状态标志位
先决条件	-
被调用函数	-
输入参数{in}	
enet_flag	ENET模块DMA标志位清除，下列参数仅可选择一个
ENET_DMA_FLAG_TS_CLR	发送状态标志位清除
ENET_DMA_FLAG_TPS_CLR	发送流程停止状态标志位清除
ENET_DMA_FLAG_TBU_CLR	发送缓冲区不可用状态标志位清除
ENET_DMA_FLAG_TJT_CLR	发送jabber超时状态标志位清除
ENET_DMA_FLAG_RO_CLR	接收溢出状态标志位清除
ENET_DMA_FLAG_TU_CLR	发送下溢状态标志位清除
ENET_DMA_FLAG_RS_CLR	接收状态标志位清除

<i>ENET_DMA_FLAG_RBU_CLR</i>	接收缓冲区不可用状态标志位清除
<i>ENET_DMA_FLAG_RPS_CLR</i>	接受流程停止状态标志位清除
<i>ENET_DMA_FLAG_RWT_CLR</i>	接收看门狗超时状态标志位清除
<i>ENET_DMA_FLAG_ET_CLR</i>	早发送状态标志位清除
<i>ENET_DMA_FLAG_FBE_CLR</i>	致命总线错误状态标志位清除
<i>ENET_DMA_FLAG_ER_CLR</i>	早接收状态标志位清除
<i>ENET_DMA_FLAG_AI_CLR</i>	异常中断汇总标志位清除
<i>ENET_DMA_FLAG_NI_CLR</i>	正常中断汇总标志位清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the specified flag bit */
enet_flag_clear(ENET_DMA_FLAG_RS_CLR);
```

函数 **enet_interrupt_enable**

函数 **enet_interrupt_enable** 描述见下表：

表 3-216. 函数 **enet_interrupt_enable**

函数名称	enet_interrupt_enable
函数原型	void enet_interrupt_enable(enet_int_enum enet_int);
功能描述	使能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-

输入参数{in}	
enet_int	ENET中断，下列参数仅可选择一个
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_TMSTIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_RGUFIM</i>	接收“好”单播帧中断屏蔽
<i>ENET_MSC_INT_TGFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_TIE</i>	发送中断使能
<i>ENET_DMA_INT_TPSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_TBUIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_TJTIE</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_ROIE</i>	接收溢出中断使能
<i>ENET_DMA_INT_TUIE</i>	发送下溢中断使能
<i>ENET_DMA_INT_RRIE</i>	接收中断使能
<i>ENET_DMA_INT_RBUIE</i>	接收缓冲区不可用中断使能

<i>ENET_DMA_INT_R</i> <i>PSIE</i>	接收流程停止中断使能
<i>ENET_DMA_INT_R</i> <i>WTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_ET</i> <i>IE</i>	早发送中断使能
<i>ENET_DMA_INT_FB</i> <i>EIE</i>	致命总线错误中断使能
<i>ENET_DMA_INT_E</i> <i>RIE</i>	早接收中断使能
<i>ENET_DMA_INT_AI</i> <i>E</i>	异常中断汇总使能
<i>ENET_DMA_INT_NI</i> <i>E</i>	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable normal interrupt summary */
enet_interrupt_enable(ENET_DMA_INT_NIE);
```

函数 enet_interrupt_disable

函数enet_interrupt_disable描述见下表：

表 3-217. 函数 enet_interrupt_disable

函数名称	enet_interrupt_disable
函数原型	void enet_interrupt_disable(enet_int_enum enet_int);
功能描述	禁能ENET模块MAC/MSC/DMA中断
先决条件	-
被调用函数	-
输入参数{in}	

enet_int	ENET中断，下列参数仅可选择一个
<i>ENET_MAC_INT_WUMIM</i>	WUM中断屏蔽
<i>ENET_MAC_INT_TMSTIM</i>	时间戳触发中断屏蔽
<i>ENET_MSC_INT_RFCEIM</i>	接收帧CRC错误中断屏蔽
<i>ENET_MSC_INT_RFAEIM</i>	接收帧对齐错误中断屏蔽
<i>ENET_MSC_INT_RGUFIM</i>	接收“好”单播帧中断屏蔽
<i>ENET_MSC_INT_TGFSCIM</i>	仅遇到1个冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFMSCIM</i>	遇到1个以上冲突后发送“好”帧中断屏蔽
<i>ENET_MSC_INT_TGFIM</i>	发送“好”的帧的中断屏蔽
<i>ENET_DMA_INT_TE</i>	发送中断使能
<i>ENET_DMA_INT_TPSIE</i>	发送流程停止中断使能
<i>ENET_DMA_INT_TBUIE</i>	发送缓冲区不可用中断使能
<i>ENET_DMA_INT_TJTIIE</i>	发送jabber超时中断使能
<i>ENET_DMA_INT_ROIE</i>	接收溢出中断使能
<i>ENET_DMA_INT_TUIE</i>	发送下溢中断使能
<i>ENET_DMA_INT_RIIE</i>	接收中断使能
<i>ENET_DMA_INT_BUIE</i>	接收缓冲区不可用中断使能
<i>ENET_DMA_INT_R</i>	接收流程停止中断使能

<i>PSIE</i>	
<i>ENET_DMA_INT_RWTIE</i>	接收看门狗超时中断使能
<i>ENET_DMA_INT_ETIE</i>	早发送中断使能
<i>ENET_DMA_INT_FBEIE</i>	致命总线错误中断使能
<i>ENET_DMA_INT_ERIE</i>	早接收中断使能
<i>ENET_DMA_INT_AI</i>	异常中断汇总使能
<i>ENET_DMA_INT_NIE</i>	正常中断汇总使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable normal interrupt summary */
enet_interrupt_disable(ENET_DMA_INT_NIE);
```

函数 enet_interrupt_flag_get

函数enet_interrupt_flag_get描述见下表：

表 3-218. 函数 enet_interrupt_flag_get

函数名称	enet_interrupt_flag_get
函数原型	FlagStatus enet_interrupt_flag_get(enet_int_flag_enum int_flag);
功能描述	获取ENET模块MAC/MSC/DMA中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	ENET中断标志位，下列参数仅可选择一个

<i>ENET_MAC_INT_FL</i> AG_WUM	WUM中断标志位
<i>ENET_MAC_INT_FL</i> AG_MSC	MSC中断标志位
<i>ENET_MAC_INT_FL</i> AG_MSCR	MSC接收中断标志位
<i>ENET_MAC_INT_FL</i> AG_MSCT	MSC发送中断标志位
<i>ENET_MAC_INT_FL</i> AG_TMST	时间戳触发中断标志位
<i>ENET_MSC_INT_FL</i> AG_RFCE	接收帧CRC错误中断标志位
<i>ENET_MSC_INT_FL</i> AG_RFAE	接收帧对齐错误中断标志位
<i>ENET_MSC_INT_FL</i> AG_RGUF	接收“好”单播帧中断标志位
<i>ENET_MSC_INT_FL</i> AG_TGFSC	仅遇到1个冲突后发送“好”帧中断标志位
<i>ENET_MSC_INT_FL</i> AG_TGFMSC	遇到1个以上冲突后发送“好”帧中断标志位
<i>ENET_MSC_INT_FL</i> AG_TGF	发送“好”的帧的中断标志位
<i>ENET_DMA_INT_FL</i> AG_TS	发送中断标志位
<i>ENET_DMA_INT_FL</i> AG_TPS	发送流程停止中断标志位
<i>ENET_DMA_INT_FL</i> AG_TBU	发送缓冲区不可用中断标志位
<i>ENET_DMA_INT_FL</i> AG_TJT	发送jabber超时中断标志位
<i>ENET_DMA_INT_FL</i> AG_RO	接收溢出中断标志位
<i>ENET_DMA_INT_FL</i> AG_TU	发送下溢中断标志位

<i>ENET_DMA_INT_FL</i> <i>AG_RS</i>	接收中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RBU</i>	接收缓冲区不可用中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RPS</i>	接收流程停止中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_RWT</i>	接收看门狗超时中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_ET</i>	早发送中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_FBE</i>	致命总线错误中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_ER</i>	早接收中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_AI</i>	异常中断汇总中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_NI</i>	正常中断汇总中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_MSC</i>	MSC中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_WUM</i>	WUM中断标志位
<i>ENET_DMA_INT_FL</i> <i>AG_TST</i>	时间戳触发中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* check whether the specified flag bit is set or not */
enet_interrupt_flag_get(ENET_DMA_INT_FLAG_RS);
```

函数 enet_interrupt_flag_clear

函数enet_interrupt_flag_clear描述见下表：

表 3-219. 函数 enet_interrupt_flag_clear

函数名称	enet_interrupt_flag_clear
函数原型	void enet_interrupt_flag_clear(enet_int_flag_clear_enum int_flag_clear);
功能描述	禁能ENET中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
int_flag_clear	ENET中断标志位清除，下列参数仅可选择一个
ENET_DMA_INT_FL AG_TS_CLR	发送中断标志位清除
ENET_DMA_INT_FL AG_TPS_CLR	发送流程停止中断标志位清除
ENET_DMA_INT_FL AG_TBU_CLR	发送缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_TJT_CLR	发送jabber超时中断标志位清除
ENET_DMA_INT_FL AG_RO_CLR	接收溢出中断标志位清除
ENET_DMA_INT_FL AG_TU_CLR	发送下溢中断标志位清除
ENET_DMA_INT_FL AG_RS_CLR	接收中断标志位清除
ENET_DMA_INT_FL AG_RBU_CLR	接收缓冲区不可用中断标志位清除
ENET_DMA_INT_FL AG_RPS_CLR	接收流程停止中断标志位清除
ENET_DMA_INT_FL AG_RWT_CLR	接收看门狗超时中断标志位清除
ENET_DMA_INT_FL AG_ET_CLR	早发送中断标志位清除

<i>ENET_DMA_INT_FL</i>	致命总线错误中断标志位清除
<i>AG_FBE_CLR</i>	
<i>ENET_DMA_INT_FL</i>	早接收中断标志位清除
<i>AG_ER_CLR</i>	
<i>ENET_DMA_INT_FL</i>	异常中断汇总中断标志位清除
<i>AG_AI_CLR</i>	
<i>ENET_DMA_INT_FL</i>	正常中断汇总中断标志位清除
<i>AG_NI_CLR</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear receive status flag bit */

enet_interrupt_flag_clear(ENET_DMA_INT_FLAG_RS);
```

函数 enet_tx_enable

函数enet_tx_enable描述见下表：

表 3-220. 函数 enet_tx_enable

函数名称	enet_tx_enable
函数原型	void enet_tx_enable(void);
功能描述	ENET发送功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable transport function of MAC and DMA */
```

```
enet_tx_enable();
```

函数 enet_tx_disable

函数enet_tx_disable描述见下表：

表 3-221. 函数 enet_tx_disable

函数名称	enet_tx_disable
函数原型	void enet_tx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	enet_txfifo_flush()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable transport function of MAC and DMA */
```

```
enet_tx_disable();
```

函数 enet_rx_enable

函数enet_rx_enable描述见下表：

表 3-222. 函数 enet_rx_enable

函数名称	enet_rx_enable
函数原型	void enet_rx_enable(void);
功能描述	ENET接收功能使能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable reception function of MAC and DMA */
enet_rx_enable();
```

函数 **enet_rx_disable**

函数enet_rx_disable描述见下表：

表 3-223. 函数 **enet_rx_disable**

函数名称	enet_rx_disable
函数原型	void enet_rx_disable(void);
功能描述	ENET发送功能禁能（包括ENET外设内的MAC和DMA模块）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable reception function of MAC and DMA */
enet_rx_disable();
```

函数 **enet_registers_get**

函数enet_registers_get描述见下表：

表 3-224. 函数 enet_registers_get

函数名称	enet_registers_get
函数原型	void enet_registers_get(enet_registers_type_enum type, uint32_t *preg, uint32_t num);
功能描述	获取指定范围ENET寄存器值
先决条件	-
被调用函数	-
输入参数{in}	
type	寄存器类型, 下列参数仅可选择一个
ALL_MAC_REG	寄存器范围从ENET_MAC_CFG到ENET_MAC_FCTH
ALL_MSC_REG	寄存器范围从ENET_MSC_CTL到ENET_MSC_RGUFHCNT
ALL_PTP_REG	寄存器范围从ENET_PTP_TSCTL到ENET_PTP_PPSCTL
ALL_DMA_REG	寄存器范围从ENET_DMA_BCTL到ENET_DMA_CRBADDR
输入参数{in}	
num	想要获取的寄存器个数, 范围(0~54)
输出参数{out}	
preg	存储寄存器值的应用缓冲区指针
返回值	
-	-

例如:

```
/* get all mac registers value */
uint32_t register_buffer[5];
enet_registers_get(ALL_MAC_REG, 5, register_buffer);
```

函数 enet_debug_status_get

函数enet_debug_status_get描述见下表:

表 3-225. 函数 enet_debug_status_get

函数名称	enet_debug_status_get
函数原型	uint32_t enet_debug_status_get(uint32_t mac_debug);
功能描述	获取ENET调试状态信息

先决条件	-
被调用函数	-
输入参数{in}	
mac_debug	ENET调试信息选项，下列参数仅可选择一个
ENET_MAC_RECEIVER_NOT_IDLE	MAC接收器非空
ENET_RX_ASYNC_RONOUS_FIFO_STATUS	异步接收FIFO状态
ENET_RXFIFO_WRITING	接收FIFO正在执行写操作
ENET_RXFIFO_READ_STATUS	读取接收FIFO操作状态
ENET_RXFIFO_STATUS	接收FIFO状态
ENET_MAC_TRANSMITTER_NOT_IDLE	MAC发送器非空
ENET_MAC_TRANSMITTER_STATUS	MAC发送器状态
ENET_PAUSE_CONDITION_STATUS	暂停条件状态
ENET_TXFIFO_READ_STATUS	读取发送FIFO操作状态
ENET_TXFIFO_WRITING	发送FIFO正在执行写操作
ENET_TXFIFO_NOT_EMPTY	发送FIFO非空
ENET_TXFIFO_FULL	发送FIFO已满
输出参数{out}	
-	-
返回值	
uint32_t	相关状态值

例如：

```
/* get debug message of RxFIFO state */

uint32_t debug_value;

debug_value = enet_debug_status_get (ENET_RXFIFO_STATE);
```

函数 **enet_address_filter_enable**

函数**enet_address_filter_enable**描述见下表：

表 3-226. 函数 **enet_address_filter_enable**

函数名称	enet_address_filter_enable
函数原型	void enet_address_filter_enable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器使能
先决条件	-
被调用函数	--
输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
ENET_MAC_ADDRESS1	使能MAC地址组1过滤器
ENET_MAC_ADDRESS2	使能MAC地址组2过滤器
ENET_MAC_ADDRESS3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the MAC address 1 filter */

enet_address_filter_enable(ENET_MAC_ADDRESS1);
```

函数 **enet_address_filter_disable**

函数**enet_address_filter_disable**描述见下表：

表 3-227. 函数 enet_address_filter_disable

函数名称	enet_address_filter_disable
函数原型	void enet_address_filter_disable(enet_macaddress_enum mac_addr);
功能描述	MAC地址过滤器禁能
先决条件	-
被调用函数	-
输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
ENET_MAC_ADDRESS1	使能MAC地址组1过滤器
ENET_MAC_ADDRESS2	使能MAC地址组2过滤器
ENET_MAC_ADDRESS3	使能MAC地址组3过滤器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the MAC address 1 filter */
enet_address_filter_disable(ENET_MAC_ADDRESS1);
```

函数 enet_address_filter_config

函数enet_address_filter_config描述见下表：

表 3-228. 函数 enet_address_filter_config

函数名称	enet_address_filter_config
函数原型	void enet_address_filter_config(enet_macaddress_enum mac_addr, uint32_t addr_mask, uint32_t filter_type);
功能描述	配置MAC地址过滤器模式
先决条件	-
被调用函数	-

输入参数{in}	
mac_addr	选择被使能的MAC地址组，下列参数仅可选择一个
<i>ENET_MAC_ADDRESS1</i>	使能MAC地址组1过滤器
<i>ENET_MAC_ADDRESS2</i>	使能MAC地址组2过滤器
<i>ENET_MAC_ADDRESS3</i>	使能MAC地址组3过滤器
输入参数{in}	
addr_mask	选择MAC地址哪些字节将被屏蔽，下列参数可以选择多个
<i>ENET_ADDRESS_MASK_BYTE0</i>	屏蔽ENET_MAC_ADDR1L[7:0] bits
<i>ENET_ADDRESS_MASK_BYTE1</i>	屏蔽ENET_MAC_ADDR1L[15:8] bits
<i>ENET_ADDRESS_MASK_BYTE2</i>	屏蔽ENET_MAC_ADDR1L[23:16] bits
<i>ENET_ADDRESS_MASK_BYTE3</i>	屏蔽ENET_MAC_ADDR1L [31:24] bits
<i>ENET_ADDRESS_MASK_BYTE4</i>	屏蔽ENET_MAC_ADDR1H [7:0] bits
<i>ENET_ADDRESS_MASK_BYTE5</i>	屏蔽ENET_MAC_ADDR1H [15:8] bits
输入参数{in}	
filter_type	选择MAC地址过滤器类型， 下列参数仅可选择一个
<i>ENET_ADDRESS_FILTER_SA</i>	过滤器比对接收帧MAC地址的源地址域
<i>ENET_ADDRESS_FILTER_DA</i>	过滤器比对接收帧MAC地址的目的地址域
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* config the MAC address 1 filter */

enet_address_filter_config(ENET_MAC_ADDRESS1, ENET_ADDRESS_MASK_BYTE0 |
ENET_ADDRESS_MASK_BYTE1 | ENET_ADDRESS_MASK_BYTE2, ENET_ADDRESS
_FILTER_DA);
```

函数 **enet_phy_config**

函数enet_phy_config描述见下表：

表 3-229. 函数 **enet_phy_config**

函数名称	enet_phy_config
函数原型	ErrStatus enet_phy_config(void);
功能描述	PHY接口配置（配置SMII时钟并复位PHY芯片）
先决条件	-
被调用函数	rcu_clock_freq_get()/enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* config PHY interface */

enet_phy_config();
```

函数 **enet_phy_write_read**

函数enet_phy_write_read描述见下表：

表 3-230. 函数 **enet_phy_write_read**

函数名称	enet_phy_write_read
函数原型	ErrStatus enet_phy_write_read(enet_phydirection_enum direction, uint16_t phy_address, uint16_t phy_reg, uint16_t *pvalue);

功能描述	写/读PHY寄存器
先决条件	-
被调用函数	-
输入参数{in}	
direction	下列参数仅可选择一个
ENET_PHY_WRITE	向PHY寄存器写数据
ENET_PHY_READ	从PHY寄存器读数据
输入参数{in}	
phy_address	0x0 - 0x1F
输入参数{in}	
phy_reg	0x0 - 0x1F
输入参数{in}	
pvalue	当direction选择ENET_PHY_WRITE时，表示将写入PHY寄存器的值
输出参数{out}	
pvalue-	当direction选择ENET_PHY_READ时，表示将存储PHY寄存器读出的值
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* write 0 to PHY BCR register */
uint16_t temp_phy = 0U;
phy_state = enet_phy_write_read(ENET_PHY_WRITE, PHY_ADDRESS, PHY_REG_BCR,
&temp_phy);
```

函数 enet_phyloopback_enable

函数enet_phyloopback_enable描述见下表：

表 3-231. 函数 enet_phyloopback_enable

函数名称	enet_phyloopback_enable
函数原型	ErrStatus enet_phyloopback_enable(void);
功能描述	使能PHY芯片回环模式

先决条件	-
被调用函数	enet_phy_write_read()
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* enable the loopback function of PHY chip */

ErrStatus phy_state = ERROR;

phy_state = enet_phyloopback_enable();
```

函数 enet_phyloopback_disable

函数enet_phyloopback_disable描述见下表：

表 3-232. 函数 enet_phyloopback_disable

函数名称	enet_phyloopback_disable
函数原型	ErrStatus enet_phyloopback_disable(void);
功能描述	禁能PHY芯片回环模式
先决条件	-
被调用函数	enet_phy_write_read
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* disable the loopback function of PHY chip */
```

```

ErrStatus phy_state = ERROR;
phy_state = enet_phyloopback_disable();

```

函数 **enet_forward_feature_enable**

函数**enet_forward_feature_enable**描述见下表:

表 3-233. 函数 **enet_forward_feature_enable**

函数名称	enet_forward_feature_enable
函数原型	void enet_forward_feature_enable(uint32_t feature);
功能描述	使能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
<i>ENET_AUTO_PADC RC_DROP</i>	自动去除接收帧填充字节和FCS域
<i>ENET_TYPEFRAME _CRC_DROP</i>	帧通过前自动去除FCS域最后四个字节的CRC校验和
<i>ENET_FORWARD_ ERRFRAMES</i>	除了过短帧外的其他错误帧都会转发给应用
<i>ENET_FORWARD_ UNDERSZ_GOODF RAMES</i>	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);

```

函数 **enet_forward_feature_disable**

函数**enet_forward_feature_disable**描述见下表:

表 3-234. 函数 enet_forward_feature_disable

函数名称	enet_forward_feature_disable
函数原型	void enet_forward_feature_disable(uint32_t feature);
功能描述	禁能ENET帧通过相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET帧通过功能，下列参数可以选择多个
ENET_AUTO_PADCRC_DROP	自动去除接收帧填充字节和FCS域
ENET_TYPEFRAME_CRC_DROP	帧通过前自动去除FCS域最后四个字节的CRC校验和
ENET_FORWARD_ERRFRAMES	除了过短帧外的其他错误帧都会转发给应用
ENET_FORWARD_UNDERSZ_GOODFRAMES	帧长小于64字节但没有错误的帧将转发给应用
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that forwarding undersized good frames */
enet_forward_feature_enable(ENET_FORWARD_UNDERSZ_GOODFRAMES);
```

函数 enet_fliter_feature_enable

函数enet_fliter_feature_enable描述见下表：

表 3-235. 函数 enet_fliter_feature_enable

函数名称	enet_fliter_feature_enable
函数原型	void enet_fliter_feature_enable(uint32_t feature)
功能描述	使能ENET帧过滤器相关功能

先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
<i>ENET_SRC_FILTER</i>	源地址过滤器
<i>ENET_SRC_FILTER_INVERSE</i>	源地址过滤结果逆转
<i>ENET_DEST_FILTER_INVERSE</i>	目的地址过滤结果逆转
<i>ENET_MULTICAST_FILTER_PASS</i>	接收多播帧
<i>ENET_MULTICAST_FILTER_HASH_MODE</i>	HASH多播过滤器
<i>ENET_UNICAST_FILTER_HASH_MODE</i>	HASH单播过滤器
<i>ENET_FILTER_MODE_EITHER</i>	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

函数 **enet_filter_feature_disable**

函数**enet_filter_feature_disable**描述见下表：

表 3-236. 函数 **enet_filter_feature_disable**

函数名称	enet_filter_feature_disable
函数原型	void enet_filter_feature_disable(uint32_t feature);

功能描述	禁能ENET帧过滤器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET过滤器功能，下列参数可以选择多个
ENET_SRC_FILTER	源地址过滤器
ENET_SRC_FILTER_INVERSE	源地址过滤结果逆转
ENET_DEST_FILTER_INVERSE	目的地址过滤结果逆转
ENET_MULTICAST_FILTER_PASS	接收多播帧
ENET_MULTICAST_FILTER_HASH_MODE	HASH多播过滤器
ENET_UNICAST_FILTER_HASH_MODE	HASH单播过滤器
ENET_FILTER_MODE_EITHER	HASH或完美过滤器功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable filter source address function */
enet_filter_feature_enable(ENET_SRC_FILTER);
```

函数 enet_pauseframe_generate

函数enet_pauseframe_generate描述见下表：

表 3-237. 函数 enet_pauseframe_generate

函数名称	enet_pauseframe_generate
------	--------------------------

函数原型	ErrStatus enet_pauseframe_generate(void);
功能描述	生成暂停帧，使能发送流控功能后ENET模块将发送暂停帧
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* generate the pause frame */
ErrStatus rval;
rval = enet_pauseframe_generate();
```

函数 enet_pauseframe_detect_config

函数enet_pauseframe_detect_config描述见下表：

表 3-238. 函数 enet_pauseframe_detect_config

函数名称	enet_pauseframe_detect_config
函数原型	void enet_pauseframe_detect_config(uint32_t detect);
功能描述	配置暂停帧检测类型
先决条件	-
被调用函数	-
输入参数{in}	
detect	暂停帧检测，下列参数仅可选择一个
<i>ENET_MAC0_AND_UNIQUE_ADDRESS_PAUSEDTECT</i>	除了唯一多播地址的暂停帧，MAC同时还会使用MAC0地址（ENET_MAC_ADDR0H寄存器和ENET_MAC_ADDR0L寄存器）来检测暂停帧
<i>ENET_UNIQUE_PAUSEDTECT</i>	MAC只接收符合IEEE802.3规范定义的唯一多播地址的暂停帧

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config the pause frame detect type as ENET_UNIQUE_PAUSEDTECT */
enet_pauseframe_detect_config(ENET_UNIQUE_PAUSEDTECT);
```

函数 enet_pauseframe_config

函数enet_pauseframe_config描述见下表：

表 3-239. 函数 enet_pauseframe_config

函数名称	enet_pauseframe_config
函数原型	void enet_pauseframe_config(uint32_t pausetime, uint32_t pause_threshold);
功能描述	配置暂停帧参数
先决条件	-
被调用函数	-
输入参数{in}	
pausetime	暂停控制帧时间域，范围(0~0xFFFF)
输入参数{in}	
pause_threshold	设置了自动重发暂停帧的定时器阈值。这个阈值应当大于0，小于位[31:16]定义的暂停时间。低阈值的计算公式为PTM-PLTS。例如，PTM = 0x80（128个时间间隙），PLTS = 0x1（28个时间间隙），那么在第一个暂停帧发出100(128-28)个时间间隙后，将自动重发第二个暂停帧，下列参数仅可选择一个
ENET_PAUSETIME_MINUS4	暂停时间 - 4个时间间隙
ENET_PAUSETIME_MINUS28	暂停时间 - 28个时间间隙
ENET_PAUSETIME_MINUS144	暂停时间 - 144个时间间隙
ENET_PAUSETIME_MINUS256	暂停时间 - 256个时间间隙

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config pause time minus 4 slot times */

enet_pauseframe_config(30, ENET_PAUSETIME_MINUS4);
```

函数 enet_flowcontrol_threshold_config

函数enet_flowcontrol_threshold_config描述见下表：

表 3-240. 函数 enet_flowcontrol_threshold_config

输入参数{in}	
deactive	流控失效的阈值。这个值应当小于位[2:0]定义的流控激活阈值。当RxFIFO中未处理的数据低于这些位所设置的值，流控功能将自动失效，下列参数仅可选择一个
<i>ENET_DEACTIVE_THRESHOLD_256BYTES</i>	256字节
<i>ENET_DEACTIVE_THRESHOLD_512BYTES</i>	512字节
<i>ENET_DEACTIVE_THRESHOLD_768BYTES</i>	768字节
<i>ENET_DEACTIVE_THRESHOLD_1024BYTES</i>	1024字节

<i>ENET_DEACTIVE_T</i>	
<i>HRESHOLD_1280B</i>	1280字节
<i>YTES</i>	
<i>ENET_DEACTIVE_T</i>	
<i>HRESHOLD_1536B</i>	1536字节
<i>YTES</i>	
<i>ENET_DEACTIVE_T</i>	
<i>HRESHOLD_1792B</i>	1792字节
<i>YTES</i>	
输入参数{in}	
active	流控激活的阈值。若使能了流控功能，当Rx FIFO中未处理的数据超过了这些位所设置的值，流控功能将被激活，下列参数仅可选择一个
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_256BYTE</i>	256字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_512BYTE</i>	512字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_768BYTE</i>	768字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_1024BYTE</i>	1024字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_1280BYTE</i>	1280字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_1536BYTE</i>	1536字节
<i>S</i>	
<i>ENET_ACTIVE_THR</i>	
<i>ESHOLD_1792BYTE</i>	1792字节
<i>S</i>	
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the threshold of the flow control */
enet_flowcontrol_threshold_config(ENET_DEACTIVE_THRESHOLD_256BYTES, ENET_ACTIVE_THRESHOLD_256BYTES);
```

函数 enet_flowcontrol_feature_enable

函数enet_flowcontrol_feature_enable描述见下表：

表 3-241. 函数 enet_flowcontrol_feature_enable

函数名称	enet_flowcontrol_feature_enable
函数原型	void enet_flowcontrol_feature_enable(uint32_t feature);
功能描述	使能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	发送流控功能
ENET_BACK_PRESSURE	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the flow control operation in the MAC */
enet_flowcontrol_feature_enable(ENET_ZERO_QUANTA_PAUSE);
```

函数 enet_flowcontrol_feature_disable

函数enet_flowcontrol_feature_disable描述见下表：

表 3-242. 函数 enet_flowcontrol_feature_disable

函数名称	enet_flowcontrol_feature_disable
函数原型	void enet_flowcontrol_feature_disable(uint32_t feature);
功能描述	禁能ENET流控相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET流控功能模式，下列参数可以选择多个
ENET_ZERO_QUANTA_PAUSE	零时间片暂停控制帧自动生成
ENET_TX_FLOWCONTROL	发送流控功能
ENET_RX_FLOWCONTROL	发送流控功能
ENET_BACK_PRESSURE	背压功能（仅在半双工模式下）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the automatic zero-quanta generation function */
enet_flowcontrol_feature_disable(ENET_ZERO_QUANTA_PAUSE);
```

函数 enet_dmaprocess_state_get

函数enet_dmaprocess_state_get描述见下表：

表 3-243. 函数 enet_dmaprocess_state_get

函数名称	enet_dmaprocess_state_get
函数原型	uint32_t enet_dmaprocess_state_get(enet_dmadiirection_enum direction);

功能描述	获取DMA发送/接收流程状态
先决条件	-
被调用函数	-
输入参数{in}	
direction	DMA传输方向
ENET_DMA_TX	DMA发送进程
ENET_DMA_RX	DMA接收进程
输出参数{out}	
-	-
返回值	
uint32_t	DMA流程状态，可取值如下： <i>ENET_RX_STATE_STOPPED / ENET_RX_STATE_FETCHING / ENET_RX_STATE_WAITING / ENET_RX_STATE_SUSPENDED / ENET_RX_STATE_CLOSING / ENET_RX_STATE_QUEUEING / ENET_TX_STATE_STOPPED / ENET_TX_STATE_FETCHING / ENET_TX_STATE_WAITING / ENET_TX_STATE_READING / ENET_TX_STATE_SUSPENDED / ENET_TX_STATE_CLOSING</i>

例如：

```

/* get the dma receive process state */

uint32_t reval;

reval = enet_dmaprocess_state_get(ENET_DMA_RX);

if(ENET_RX_STATE_SUSPENDED == reval){

    do...

}

```

函数 **enet_dmaprocess_resume**

函数**enet_dmaprocess_resume**描述见下表：

表 3-244. 函数 *enet_dmaprocess_resume*

函数名称	enet_dmaprocess_resume
函数原型	void enet_dmaprocess_resume(enet_dmadirection_enum direction);
功能描述	DMA发送/接收查询使能

先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA发送进程
ENET_DMA_RX	DMA接收进程
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable DMA receive process */

enet_dmaprocess_resume(ENET_DMA_RX);
```

函数 enet_rxprocess_check_recovery

函数enet_rxprocess_check_recovery描述见下表：

表 3-245. 函数 enet_rxprocess_check_recovery

函数名称	enet_rxprocess_check_recovery
函数原型	void enet_rxprocess_check_recovery(void);
功能描述	检测并恢复接收流程
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* check and recover the Rx process */
```

```
enet_rxprocess_check_recovery();
```

函数 enet_txfifo_flush

函数enet_txfifo_flush描述见下表:

表 3-246. 函数 enet_txfifo_flush

函数名称	enet_txfifo_flush
函数原型	ErrStatus enet_txfifo_flush(void);
功能描述	刷新ENET发送FIFO，并等待刷新操作完成
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* flush the ENET transmit FIFO */
ErrStatus reval = ERROR;
reval = enet_txfifo_flush();
```

函数 enet_current_desc_address_get

函数enet_current_desc_address_get描述见下表:

表 3-247. 函数 enet_current_desc_address_get

函数名称	enet_current_desc_address_get
函数原型	uint32_t enet_current_desc_address_get(enet_desc_reg_enum addr_get);
功能描述	获取当前发送/接收描述符地址、当前缓冲区地址、描述符列表首地址
先决条件	-
被调用函数	-

输入参数{in}	
addr_get	可获取的描述符地址类型，下列参数仅可选择一个
<i>ENET_RX_DESC_TABLE</i>	接收描述符列表首地址
<i>ENET_RX_CURRENT_DESC</i>	当前DMA控制器使用的接收描述符地址
<i>ENET_RX_CURRENT_BUFFER</i>	当前DMA控制器使用的接收描述符缓冲区地址
<i>ENET_TX_DESC_TABLE</i>	发送描述符列表首地址
<i>ENET_TX_CURRENT_DESC</i>	当前DMA控制器使用的发送描述符地址
<i>ENET_TX_CURRENT_BUFFER</i>	当前DMA控制器使用的发送描述符缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	0- 0xFFFFFFFF

例如：

```
/* get the start address of the receive descriptor table */
uint32_t reval;
reval = enet_current_desc_address_get(ENET_RX_DESC_TABLE);
```

函数 **enet_desc_information_get**

函数**enet_desc_information_get**描述见下表：

表 3-248. 函数 **enet_desc_information_get**

函数名称	enet_desc_information_get
函数原型	uint32_t enet_desc_information_get(enet_descriptors_struct *desc, enetcstate_enum info_get);
功能描述	获取发送/接收描述符详细信息
先决条件	-

被调用函数	-
输入参数{in}	
desc	描述符指针, 结构体成员介绍请参考 结构体enet_descriptors_struct
输入参数{in}	
info_get	可选择的描述符信息类型, 下列参数仅可选择一个
RXDESC_BUFFER_1_SIZE	接收缓冲区1大小
RXDESC_BUFFER_2_SIZE	接收缓冲区2大小
RXDESC_FRAME_LENGTH	接收帧长度
TXDESC_COLLISION_N_COUNT	帧发送出去前出现的冲突次数
RXDESC_BUFFER_1_ADDR	接收帧的缓冲区地址
TXDESC_BUFFER_1_ADDR	发送帧的缓冲区地址
输出参数{out}	
-	-
返回值	
uint32_t	描述符信息, 如果返回值为0xFFFFFFFFU, 说明输入参数有误

例如:

```
/* get the reception buffer 1 size */
uint32_t reval;
reval = enet_desc_information_get(rx_desc, RXDESC_BUFFER_1_SIZE);
```

函数 enet_missed_frame_counter_get

函数enet_missed_frame_counter_get描述见下表:

表 3-249. 函数 enet_missed_frame_counter_get

函数名称	enet_missed_frame_counter_get
函数原型	void enet_missed_frame_counter_get(uint32_t *rxfifo_drop, uint32_t

	*rxdma_drop);
功能描述	获取接收丢弃帧数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
rxfifo_drop	存储由于过短帧或接收FIFO溢出而丢弃帧数的指针
输出参数{out}	
rxdma_drop	存储由于接收描述符不可用而丢弃帧数的指针
返回值	
-	-

例如：

```
/* get the number of missed frames during receiving */
uint32_t rxcnt, txcnt;
enet_missed_frame_counter_get(&rxcnt, &txcnt);
```

函数 enet_desc_flag_get

函数enet_desc_flag_get描述见下表：

表 3-250. 函数 enet_desc_flag_get

函数名称	enet_desc_flag_get
函数原型	FlagStatus enet_desc_flag_get(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	获取ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针, 结构体成员介绍请参考 结构体enet_descriptors_struct
输入参数{in}	
desc_flag	描述符标志位, 下列参数仅可选择一个

(the value according to the parameter desc)	
当 desc 参数为发送描述符时	
<i>ENET_TDES0_DB</i>	顺延位
<i>ENET_TDES0_UFE</i>	数据下溢错误位
<i>ENET_TDES0_EXD</i>	过度顺延位
<i>ENET_TDES0_VFR_M</i>	VLAN帧位
<i>ENET_TDES0_ECO</i>	过度冲突位
<i>ENET_TDES0_LCO</i>	延迟冲突位
<i>ENET_TDES0_NCA</i>	无载波位
<i>ENET_TDES0_LCA</i>	载波丢失位
<i>ENET_TDES0_IPPE</i>	IP数据错误位
<i>ENET_TDES0_FRM_F</i>	帧清空位
<i>ENET_TDES0_JT</i>	Jabber超时位
<i>ENET_TDES0_ES</i>	错误汇总
<i>ENET_TDES0_IPHE</i>	IP报头错误位
<i>ENET_TDES0_TTM_SS</i>	发送时间戳状态位
<i>ENET_TDES0_TCH_M</i>	第二地址链表模式位
<i>ENET_TDES0_TER_M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS_EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA_D</i>	不填充位
<i>ENET_TDES0_DCR_C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位

<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 desc 参数为接收描述符时	
<i>ENET_RDES0_PCE</i> <i>RR</i>	数据校验和错误
<i>ENET_RDES0_CER</i> <i>R</i>	CRC错误
<i>ENET_RDES0_DBE</i> <i>RR</i>	Dribble位错误
<i>ENET_RDES0_RER</i> <i>R</i>	接收错误
<i>ENET_RDES0_RWD</i> <i>T</i>	接收看门狗超时
<i>ENET_RDES0_FRM</i> <i>T</i>	帧类型
<i>ENET_RDES0_LCO</i>	延迟冲突位
<i>ENET_RDES0_IPHE</i> <i>RR</i>	IP帧报头校验和错误
<i>ENET_RDES0_LDE</i> <i>S</i>	最后一个描述符
<i>ENET_RDES0_FDE</i> <i>S</i>	第一个描述符
<i>ENET_RDES0_VTA</i> <i>G</i>	VLAN标签位
<i>ENET_RDES0_OER</i> <i>R</i>	溢出错误位
<i>ENET_RDES0_LER</i> <i>R</i>	长度错误位
<i>ENET_RDES0_SAF</i> <i>F</i>	未通过源地址过滤器位
<i>ENET_RDES0_DER</i> <i>R</i>	描述符错误位

<i>ENET_RDES0_ERR_S</i>	错误汇总位
<i>ENET_RDES0_DAF_F</i>	未通过目标地址过滤器位
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* get the bit flag of ENET DMA descriptor */
FlagStatus reval;
reval = enet_desc_flag_get(p_txdesc, ENET_TDES0_TCHM);
```

函数 **enet_desc_flag_set**

函数**enet_desc_flag_set**描述见下表：

表 3-251. 函数 **enet_desc_flag_set**

函数名称	enet_desc_flag_set
函数原型	void enet_desc_flag_set(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	设置ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 结构体enet_descriptors_struct
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位，下列参数仅可选择一个
当 desc 参数为发送描述符时	

<i>ENET_TDES0_VFR</i> <i>M</i>	VLAN帧位
<i>ENET_TDES0_FRM</i> <i>F</i>	帧清空位
<i>ENET_TDES0_TCH</i> <i>M</i>	第二地址链表模式位
<i>ENET_TDES0_TER</i> <i>M</i>	环形发送结束模式位
<i>ENET_TDES0_TTS</i> <i>EN</i>	使能发送时间戳位
<i>ENET_TDES0_DPA</i> <i>D</i>	不填充位
<i>ENET_TDES0_DCR</i> <i>C</i>	不计算CRC位
<i>ENET_TDES0_FSG</i>	第一分块位
<i>ENET_TDES0_LSG</i>	最后分块位
<i>ENET_TDES0_INTC</i>	完成时中断位
<i>ENET_TDES0_DAV</i>	DAV位
当 <i>desc</i> 参数为接收描述符时	
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set VLAN frame bit flag of ENET DMA descriptor */
enet_desc_flag_set(p_txdesc, ENET_TDES0_VFRM);
```

函数 **enet_desc_flag_clear**

函数**enet_desc_flag_clear**描述见下表：

表 3-252. 函数 enet_desc_flag_clear

函数名称	enet_desc_flag_clear
函数原型	void enet_desc_flag_clear(enet_descriptors_struct *desc, uint32_t desc_flag);
功能描述	清除ENET模块DMA描述符标志位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针, 结构体成员介绍请参考 结构体enet_descriptors_struct
输入参数{in}	
desc_flag (the value according to the parameter desc)	描述符标志位, 下列参数仅可选择一个
当desc参数为发送描述符时	
ENET_TDES0_VFR M	VLAN帧位
ENET_TDES0_FRM F	帧清空位
ENET_TDES0_TCH M	第二地址链表模式位
ENET_TDES0_TER M	环形发送结束模式位
ENET_TDES0_TTS EN	使能发送时间戳位
ENET_TDES0_DPA D	不填充位
ENET_TDES0_DCR C	不计算CRC位
ENET_TDES0_FSG	第一分块位
ENET_TDES0_LSG	最后分块位
ENET_TDES0_INTC	完成时中断位
ENET_TDES0_DAV	DAV位

当 desc 参数为接收描述符时	
<i>ENET_RDES0_DAV</i>	描述符可用位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear VLAN frame bit flag of ENET DMA descriptor */
enet_desc_flag_clear(p_txdesc, ENET_TDES0_VFRM);
```

函数 **enet_rx_desc_immediate_receive_complete_interrupt**

函数**enet_rx_desc_immediate_receive_complete_interrupt**描述见下表：

表 3-253. 函数 **enet_rx_desc_immediate_receive_complete_interrupt**

函数名称	enet_rx_desc_immediate_receive_complete_interrupt
函数原型	void enet_rx_desc_immediate_receive_complete_interrupt(enet_descriptors_struct *desc);
功能描述	当接收完成时，立即置位ENET_DMA_STAT寄存器的RS位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 结构体enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set RS bit in ENET_DMA_STAT register immediately when receiving completed */
enet_rx_desc_immediate_receive_complete_interrupt(p_rxdesc);
```

函数 enet_rx_desc_delay_receive_complete_interrupt

函数enet_rx_desc_delay_receive_complete_interrupt描述见下表:

表 3-254. 函数 enet_rx_desc_delay_receive_complete_interrupt

函数名称	enet_rx_desc_delay_receive_complete_interrupt
函数原型	void enet_rx_desc_delay_receive_complete_interrupt(enet_descriptors_struct *desc, uint32_t delay_time);
功能描述	当接收完成时，延迟指定时间再置位ENET_DMA_STAT寄存器的RS位
先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针，结构体成员介绍请参考 结构体enet_descriptors_struct
输入参数{in}	
delay_time	延迟时间，实际延迟时间为256*delay_time个HCLK (0~0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* when receiving completed, RS bit in ENET_DMA_STAT register will be set after 256*16
HCLK */

enet_rx_desc_delay_receive_complete_interrupt(p_rxdesc, 0x00000010);
```

函数 enet_rxframe_drop

函数enet_rxframe_drop描述见下表:

表 3-255. 函数 enet_rxframe_drop

函数名称	enet_rxframe_drop
函数原型	void enet_rxframe_drop(void);
功能描述	丢弃当前接收到的帧
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* drop current receive frame */

enet_rxframe_drop();
```

函数 **enet_dma_feature_enable**

函数 **enet_dma_feature_enable** 描述见下表：

表 3-256. 函数 **enet_dma_feature_enable**

函数名称	enet_dma_feature_enable
函数原型	void enet_dma_feature_enable(uint32_t feature);
功能描述	使能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	DMA功能，下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时，RxDMA控制器清空接收帧功能
ENET_SECONDFRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable RxDMA does not flushes frames function */
```

```
enet_dma_feature_enable(ENET_NO_FLUSH_RXFRAME);
```

函数 enet_dma_feature_disable

函数enet_dma_feature_disable描述见下表:

表 3-257. 函数 enet_dma_feature_disable

函数名称	enet_dma_feature_disable
函数原型	void enet_dma_feature_disable(uint32_t feature);
功能描述	禁能ENET模块DMA相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	DMA功能, 下列参数可以选择多个
ENET_NO_FLUSH_RXFRAME	描述符不可用时, RxDMA控制器清空接收帧功能
ENET_SECOND_FRAME_OPT	TxDMA控制器第二帧功能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable RxDMA does not flushes frames function */
enet_dma_feature_disable(ENET_NO_FLUSH_RXFRAME);
```

函数 enet_rx_desc_enhanced_status_get

函数enet_rx_desc_enhanced_status_get描述见下表:

表 3-258. 函数 enet_rx_desc_enhanced_status_get

函数名称	enet_rx_desc_enhanced_status_get
函数原型	uint32_t enet_rx_desc_enhanced_status_get(enet_descriptors_struct *desc, uint32_t desc_status);
功能描述	获取接收描述符增强状态标志位信息

先决条件	-
被调用函数	-
输入参数{in}	
desc	描述符指针, 结构体成员介绍请参考 结构体enet descriptors struct
输入参数{in}	
desc_status	想要获取的状态信息
ENET_RDES4_IPPLDT	IP帧有效载荷
ENET_RDES4_IPHERR	IP帧头错误
ENET_RDES4_IPPLDERR	IP帧有效载荷错误
ENET_RDES4_IPCKSB	IP帧旁路校验和
ENET_RDES4_IPF4	Ipv4帧
ENET_RDES4_IPF6	Ipv6帧
ENET_RDES4_PTPMT	PTP消息类型
ENET_RDES4_PTOEFOEF	PTP网络帧
ENET_RDES4_PTPVF	PTP版本格式
输出参数{out}	
-	-
返回值	
uint32_t	相应的状态值

例如：

```
/* get the IP frame payload type in ENET DMA descriptor */
uint32_t status;

status = enet_rx_desc_enhanced_status_get(p_rxdesc, ENET_RDES4_IPPLDT);
```

函数 enet_desc_select_enhanced_mode

函数enet_desc_select_enhanced_mode描述见下表:

表 3-259. 函数 enet_desc_select_enhanced_mode

函数名称	enet_desc_select_enhanced_mode
函数原型	void enet_desc_select_enhanced_mode(void);
功能描述	配置DMA描述符为增强型描述符
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure descriptor to work in enhanced mode */
enet_desc_select_enhanced_mode();
```

函数 enet_ptp_enhanced_descriptors_chain_init

函数enet_ptp_enhanced_descriptors_chain_init描述见下表:

表 3-260. 函数 enet_ptp_enhanced_descriptors_chain_init

函数名称	enet_ptp_enhanced_descriptors_chain_init
函数原型	void enet_ptp_enhanced_descriptors_chain_init(enet_dmadirection_enum direction);
功能描述	初始化具有PTP功能的增强型DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型, 下列参数仅可选择一个

<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Tx descriptors's parameters in enhanced chain mode with ptp function */
enet_ptp_enhanced_descriptors_chain_init(ENET_DMA_TX);
```

函数 **enet_ptp_enhanced_descriptors_ring_init**

函数 `enet_ptp_enhanced_descriptors_ring_init` 描述见下表：

表 3-261. 函数 **enet_ptp_enhanced_descriptors_ring_init**

函数名称	<code>enet_ptp_enhanced_descriptors_ring_init</code>
函数原型	<code>void enet_ptp_enhanced_descriptors_ring_init(enet_dmadirection_enum direction);</code>
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in enhanced ring mode with ptp function */
```

```
enet_ptp_enhanced_descriptors_ring_init(ENET_DMA_RX);
```

函数 enet_ptpframe_receive_enhanced_mode

函数enet_ptpframe_receive_enhanced_mode描述见下表:

表 3-262. 函数 enet_ptpframe_receive_enhanced_mode

函数名称	enet_ptpframe_receive_enhanced_mode
函数原型	ErrStatus enet_ptpframe_receive_enhanced_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
功能描述	在PTP模式下处理当前接收到的帧，并将当前增强型描述符中存储的接收帧数据和时间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区大小
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
输出参数{out}	
timestamp	存放时间戳指针
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* receive a packet data with timestamp values to application buffer in DMA enhanced mode
*/
uint32_t rx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;
status = enet_ptpframe_receive_enhanced_mode (rx_buffer, 500, time_stamp);
```

函数 enet_ptpframe_transmit_enhanced_mode

函数enet_ptpframe_transmit_enhanced_mode描述见下表:

表 3-263. 函数 enet_ptpframe_transmit_enhanced_mode

函数名称	enet_ptpframe_transmit_enhanced_mode
函数原型	ErrStatus enet_ptpframe_transmit_enhanced_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将制定区域内的数据拷贝到当前增强型发送描述符中，并同时间戳一起发送
先决条件	-
被调用函数	-
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	SUCCESS or ERROR

例如：

```
/* send data and timestamp values in application buffer as a transmit packet with DMA
enhanced mode */

uint32_t tx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;

status = enet_ptpframe_transmit_enhanced_mode(tx_buffer, 500, time_stamp);
```

函数 enet_desc_select_normal_mode

函数enet_desc_select_normal_mode描述见下表：

表 3-264. 函数 enet_desc_select_normal_mode

函数名称	enet_desc_select_normal_mode
函数原型	void enet_desc_select_normal_mode(void);
功能描述	配置DMA描述符为常规型描述符

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure descriptor to work in normal mode */
enet_desc_select_normal_mode( );
```

函数 `enet_ptp_normal_descriptors_chain_init`

函数 `enet_ptp_normal_descriptors_chain_init` 描述见下表：

表 3-265. 函数 `enet_ptp_normal_descriptors_chain_init`

函数名称	enet_ptp_normal_descriptors_chain_init
函数原型	void enet_ptp_normal_descriptors_chain_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为链模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
<i>ENET_DMA_TX</i>	DMA Tx描述符
<i>ENET_DMA_RX</i>	DMA Rx描述符
输入参数{in}	
desc_ptptab	描述符指针，结构体成员介绍请参考 结构体enet_descriptors_struct
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal chain mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_chain_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

函数 enet_ptp_normal_descriptors_ring_init

函数enet_ptp_normal_descriptors_ring_init描述见下表：

表 3-266. 函数 enet_ptp_normal_descriptors_ring_init

函数名称	enet_ptp_normal_descriptors_ring_init
函数原型	void enet_ptp_normal_descriptors_ring_init(enet_dmadirection_enum direction, enet_descriptors_struct *desc_ptptab);
功能描述	初始化具有PTP功能的DMA接收/发送描述符为环模式
先决条件	-
被调用函数	-
输入参数{in}	
direction	描述符类型，下列参数仅可选择一个
ENET_DMA_TX	DMA Tx描述符
ENET_DMA_RX	DMA Rx描述符
输入参数{in}	
desc_ptptab	描述符指针，结构体成员介绍请参考 结构体enet_descriptors_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the DMA Rx descriptors's parameters in normal ring mode with PTP function */
enet_descriptors_struct ptp_rxdesc_tab[ENET_RXBUF_NUM];
enet_ptp_normal_descriptors_ring_init(ENET_DMA_RX, ptp_rxdesc_tab);
```

函数 enet_ptpframe_receive_normal_mode

函数enet_ptpframe_receive_normal_mode描述见下表:

表 3-267. 函数 enet_ptpframe_receive_normal_mode

函数名称	enet_ptpframe_receive_normal_mode
函数原型	ErrStatus enet_ptpframe_receive_normal_mode(uint8_t *buffer, uint32_t bufsize, uint32_t timestamp[]);
功能描述	在PTP模式下处理当前接收到的帧，并将当前描述符中存储的接收帧数据和时间戳拷贝到指定区域
先决条件	-
被调用函数	-
输入参数{in}	
bufsize	缓冲区大小
输出参数{out}	
timestamp	存放时间戳指针
输出参数{out}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到自己指定的位置
返回值	
ErrStatus	ERROR or SUCCESS

例如:

```
/* receive a packet data with timestamp values to application buffer in DMA normal mode */
uint32_t rx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;
status = enet_ptpframe_receive_normal_mode (rx_buffer, 500, time_stamp);
```

函数 enet_ptpframe_transmit_normal_mode

函数enet_ptpframe_transmit_normal_mode描述见下表:

表 3-268. 函数 enet_ptpframe_transmit_normal_mode

函数名称	enet_ptpframe_transmit_normal_mode
------	------------------------------------

函数原型	ErrStatus enet_ptpframe_transmit_normal_mode(uint8_t *buffer, uint32_t length, uint32_t timestamp[]);
功能描述	在PTP模式下将指定区域内的数据拷贝到当前发送描述符中，并同时间戳一起发送
先决条件	-
被调用函数	--
输入参数{in}	
buffer	存放数据的用户缓冲区指针，如果输入NULL，用户需要在调用该函数之前将数据拷贝到指定的位置
输入参数{in}	
length	发送数据大小
输出参数{out}	
timestamp	存放时间戳的指针，如果输入为NULL，则忽略时间戳
返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* send data and timestamp values in application buffer as a transmit packet with DMA normal mode */

uint32_t tx_buffer[500];
uint32_t time_stamp[2];
ErrStatus status;

status = enet_ptpframe_transmit_normal_mode(tx_buffer, 500, time_stamp);
```

函数 enet_wum_filter_register_pointer_reset

函数enet_wum_filter_register_pointer_reset描述见下表：

表 3-269. 函数 enet_wum_filter_register_pointer_reset

函数名称	enet_wum_filter_register_pointer_reset
函数原型	void enet_wum_filter_register_pointer_reset(void);
功能描述	远程唤醒帧过滤器寄存器指针复位
先决条件	-
被调用函数	-

输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset wakeup frame filter register pointer */

enet_wum_filter_register_pointer_reset();
```

函数 **enet_wum_filter_config**

函数enet_wum_filter_config描述见下表：

表 3-270. 函数 **enet_wum_filter_config**

函数名称	enet_wum_filter_config
函数原型	void enet_wum_filter_config(uint32_t pdata[]);
功能描述	配置远程唤醒帧寄存器
先决条件	-
被调用函数	-
输入参数{in}	
pdata	存放将写入远程唤醒帧寄存器组的数据指针（总共8字节）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the remote wakeup frame registers */

uint32_t wum_data[8];

enet_wum_filter_config (wum_data);
```

函数 enet_wum_feature_enable

函数enet_wum_feature_enable描述见下表:

表 3-271. 函数 enet_wum_feature_enable

函数名称	enet_wum_feature_enable
函数原型	void enet_wum_feature_enable(uint32_t feature);
功能描述	使能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_WUM_POWER_DOWN	掉电模式
ENET_WUM_MAGIC_PACKET_FRAME	使能接收到魔法帧的唤醒事件
ENET_WUM_WAKE_UP_FRAME	使能接收到唤醒帧的唤醒事件
ENET_WUM_GLOBAL_UNICAST	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable power down mode */
enet_wum_feature_enable(ENET_WUM_POWER_DOWN);
```

函数 enet_wum_feature_disable

函数enet_wum_feature_disable描述见下表:

表 3-272. 函数 enet_wum_feature_disable

函数名称	enet_wum_feature_disable
函数原型	void enet_wum_feature_disable(uint32_t feature)

功能描述	禁能ENET模块唤醒管理相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
<i>ENET_WUM_MAGI C_PACKET_FRAME</i>	使能接收到魔法帧的唤醒事件
<i>ENET_WUM_WAKE _UP_FRAME</i>	使能接收到唤醒帧的唤醒事件
<i>ENET_WUM_GLOB AL_UNICAST</i>	任何通过过滤器的单播帧均作为唤醒帧
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable power down mode */
enet_wum_feature_disable(ENET_WUM_POWER_DOWN);
```

函数 **enet_msc_counters_reset**

函数**enet_msc_counters_reset**描述见下表：

表 3-273. 函数 **enet_msc_counters_reset**

函数名称	enet_msc_counters_reset
函数原型	void enet_msc_counters_reset(void)
功能描述	复位MAC统计计数器组
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* reset the MAC statistics counters */
enet_msc_counters_reset();
```

函数 enet_msc_feature_enable

函数enet_msc_feature_enable描述见下表：

表 3-274. 函数 enet_msc_feature_enable

函数名称	enet_msc_feature_enable
函数原型	void enet_msc_feature_enable(uint32_t feature);
功能描述	使能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUNTER_STOP_ROLLOVER	计数器停止回转
ENET_MSC_RESET_ON_READ	读时复位
ENET_MSC_COUNTER_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable counter stop rollover function */
enet_msc_feature_enable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

函数 enet_msc_feature_disable

函数enet_msc_feature_disable描述见下表:

表 3-275. 函数 enet_msc_feature_disable

函数名称	enet_msc_feature_disable
函数原型	void enet_msc_feature_disable(uint32_t feature);
功能描述	禁能MAC统计计数器相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	下列参数可以选择多个
ENET_MSC_COUN TER_STOP_ROLLO VER	计数器停止回转
ENET_MSC_RESET _ON_READ	读时复位
ENET_MSC_COUN TERS_FREEZE	MSC计数器冻结位
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable counter stop rollover function */
enet_msc_feature_disable(ENET_MSC_COUNTER_STOP_ROLLOVER);
```

函数 enet_msc_counters_preset_config

函数enet_msc_counters_preset_config描述见下表:

表 3-276. 函数 enet_msc_counters_preset_config

函数名称	enet_msc_counters_preset_config
函数原型	void enet_msc_counters_preset_config(enet_msc_preset_enum mode);
功能描述	配置MAC统计计数器的预设模式

先决条件	-
被调用函数	-
输入参数{in}	
mode	下列参数可以选择多个
<i>ENET_MSC_PRESET_NONE</i>	关闭MSC计数器预设功能
<i>ENET_MSC_PRESET_HALF</i>	预设为最大值一半
<i>ENET_MSC_PRESET_FULL</i>	预设为最大值一半
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* preset all MSC counters to almost-half */
enet_msc_counters_preset_config(ENET_MSC_PRESET_HALF);
```

函数 **enet_msc_counters_get**

函数 **enet_msc_counters_get** 描述见下表：

表 3-277. 函数 **enet_msc_counters_get**

函数名称	enet_msc_counters_get
函数原型	uint32_t enet_msc_counters_get(enet_msc_counter_enum counter);
功能描述	获取MAC相关统计计数器值
先决条件	-
被调用函数	-
输入参数{in}	
counter	MSC计数器，下列参数仅可选择一个
<i>ENET_MSC_TX_SC_CNT</i>	MSC 1次冲突后发送“好”帧的计数器

<i>ENET_MSC_TX_MS_CCNT</i>	MSC 1次以上冲突后发送“好”帧的计数器
<i>ENET_MSC_TX_TG_FCNT</i>	MSC发送“好”帧计数器
<i>ENET_MSC_RX_RF_CECNT</i>	MSC CRC错误接收帧计数器
<i>ENET_MSC_RX_RF_AECNT</i>	MSC对齐错误接收帧计数器
<i>ENET_MSC_RX_RG_UFCNT</i>	MSC“好”单播帧接收帧计数器
输出参数{out}	
-	-
返回值	
uint32_t	MSC计数器值

例如：

```
/* get MSC transmitted good frames after a single collision counter value*/
uint32_t reval;
reval = enet_msc_counters_get(ENET_MSC_TX_SCCNT);
```

函数 **enet_ptp_subsecond_2_nanosecond**

函数**enet_ptp_subsecond_2_nanosecond**描述见下表：

表 3-278. 函数 **enet_ptp_subsecond_2_nanosecond**

函数名称	enet_ptp_subsecond_2_nanosecond
函数原型	uint32_t enet_ptp_subsecond_2_nanosecond(uint32_t subsecond);
功能描述	将亚秒值变为纳秒值
先决条件	-
被调用函数	-
输入参数{in}	
subsecond	亚秒值
输出参数{out}	
-	-

返回值	
uint32_t	纳秒值

例如：

```
/* change subsecond to nanosecond */

uin32_t reval;

reval = enet_ptp_subsecond_2_nanosecond (2);
```

函数 **enet_ptp_nanosecond_2_subsecond**

函数enet_ptp_nanosecond_2_subsecond描述见下表：

表 3-279. 函数**enet_ptp_nanosecond_2_subsecond**

函数名称	enet_ptp_nanosecond_2_subsecond
函数原型	uint32_t enet_ptp_nanosecond_2_subsecond(uint32_t nanosecond);
功能描述	将纳秒值变为亚秒值
先决条件	-
被调用函数	-
输入参数{in}	
nanosecond	纳秒值
输出参数{out}	
-	-
返回值	
uint32_t	亚秒值

例如：

```
/* change nanosecond to subsecond */

uin32_t reval;

reval = enet_ptp_nanosecond_2_subsecond (2);
```

函数 **enet_ptp_feature_enable**

函数enet_ptp_feature_enable描述见下表：

表 3-280. 函数**enet_ptp_feature_enable**

函数名称	enet_ptp_feature_enable
------	-------------------------

函数原型	void enet_ptp_feature_enable(uint32_t feature);
功能描述	使能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RXTX_TIMES_TAMP	发送/接收帧时间戳
ENET_PTP_TIMESTAMP_INT	时间戳中断触发
ENET_ALL_RX_TIMESTAMP	所有接收帧时间戳快照使能
ENET_NONTYPE_FRAME_SNAPSHOT	接收以太网帧时时间戳使能
ENET_IPV6_FRAME_SNAPSHOT	接收Ipv6帧时时间戳使能
ENET_IPV4_FRAME_SNAPSHOT	接收Ipv4帧时时间戳使能
ENET_PTP_FRAME_USE_MACADDRESS_FILTER	PTP帧MAC地址过滤使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable PTP function for all received frames */
enet_ptp_feature_enable(ENET_ALL_RX_TIMESTAMP);
```

函数 **enet_ptp_feature_disable**

函数**enet_ptp_feature_disable**描述见下表：

表 3-281. 函数 enet_ptp_feature_disable

函数名称	enet_ptp_feature_disable
函数原型	void enet_ptp_feature_disable(uint32_t feature);
功能描述	禁能PTP相关功能
先决条件	-
被调用函数	-
输入参数{in}	
feature	ENET模块PTP功能，下列参数可以选择多个
ENET_RXTX_TIMES_TAMP	发送/接收帧时间戳
ENET_PTP_TIMESTAMP_INT	时间戳中断触发
ENET_ALL_RX_TIMESTAMP	所有接收帧时间戳快照使能
ENET_NONTYPE_FRAME_SNAPSHOT	接收以太网帧时时间戳使能
ENET_IPV6_FRAME_SNAPSHOT	接收Ipv6帧时时间戳使能
ENET_IPV4_FRAME_SNAPSHOT	接收Ipv4帧时时间戳使能
ENET_PTP_FRAME_USE_MACADDRESS_FILTER	PTP帧MAC地址过滤使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PTP function for all received frames */

enet_ptp_feature_disable(ENET_ALL_RX_TIMESTAMP);
```

函数 enet_ptp_timestamp_function_config

函数enet_ptp_timestamp_function_config描述见下表:

表 3-282. 函数 enet_ptp_timestamp_function_config

函数名称	enet_ptp_timestamp_function_config
函数原型	ErrStatus enet_ptp_timestamp_function_config(enet_ptp_function_enum func);
功能描述	配置PTP时间戳相关功能
先决条件	-
被调用函数	-
输入参数{in}	
func	下列参数仅可选择一个
ENET_CKNT_ORDINARY	时间戳时钟节点类型为普通时钟
ENET_CKNT_BOUNDARY	时间戳时钟节点类型为边界时钟
ENET_CKNT_END_TO_END	时间戳时钟节点类型为端对端透明时钟
ENET_CKNT_PEER_TO_PEER	时间戳时钟节点类型为点对点透明时钟
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
ENET_PTP_FINEMODE	精调模式更新系统时间戳
ENET_PTP_COARSEMODE	粗调模式更新系统时间戳
ENET_SUBSECONDD_DIGITAL_ROLLOVER	十进制回转模式
ENET_SUBSECONDD_BINARY_ROLLOVER	二进制回转模式

VER	
<i>ENET_SNOOPING_PTP_VERSION_2</i>	监听PTP帧版本2
<i>ENET_SNOOPING_PTP_VERSION_1</i>	监听PTP帧版本1
<i>ENET_EVENT_TYPE_MESSAGES_SNA_PSHOT</i>	只接收事件类型消息使能时间戳
<i>ENET_ALL_TYPE_MESSAGES_SNAPSHOT</i>	接收到除了Announce, Management和Signaling以外的所有其他类型的消息时, 时间戳快照使能
<i>ENET_MASTER_NODE_MESSAGE_SNAPSHOT</i>	主节点消息时间戳快照使能
<i>ENET_SLAVE_NODE_MESSAGE_SNAPSHOT</i>	从节点消息时间戳快照使能
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS or ERROR

例如:

```
/* config addend register update function */

ErrStatus rval = ERROR;

rval = enet_ptp_timestamp_function_config(ENET_PTP_ADDEND_UPDATE);
```

函数 **enet_ptp_subsecond_increment_config**

函数**enet_ptp_subsecond_increment_config**描述见下表:

表 3-283. 函数 **enet_ptp_subsecond_increment_config**

函数名称	enet_ptp_subsecond_increment_config
函数原型	void enet_ptp_subsecond_increment_config(uint32_t subsecond);
功能描述	配置PTP系统时间亚秒增加值
先决条件	-

被调用函数	-
输入参数{in}	
subsecond	该值将被加到系统时间的亚秒值，范围(0~0xFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure 0x1F as system time subsecond increment value */
enet_ptp_subsecond_increment_config(0x1F);
```

函数 **enet_ptp_timestamp_addend_config**

函数 **enet_ptp_timestamp_addend_config** 描述见下表：

表 3-284. 函数 **enet_ptp_timestamp_addend_config**

函数名称	enet_ptp_timestamp_addend_config
函数原型	void enet_ptp_timestamp_addend_config(uint32_t add);
功能描述	精调模式下PTP时钟频率校准配置
先决条件	-
被调用函数	-
输入参数{in}	
add	通过将该值加到累加器用于时间同步，范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* added 0x1FFF to the accumulator register */
enet_ptp_timestamp_addend_config(0x1FFF);
```

函数 enet_ptp_timestamp_update_config

函数enet_ptp_timestamp_update_config描述见下表:

表 3-285. 函数 enet_ptp_timestamp_update_config

函数名称	enet_ptp_timestamp_update_config
函数原型	void enet_ptp_timestamp_update_config(uint32_t sign, uint32_t second, uint32_t subsecond);
功能描述	初始化时用于替换系统时间，在更新时表示在系统时间上加上或减去的秒值
先决条件	-
被调用函数	-
输入参数{in}	
sign	时间戳更新正或负符号位，下列参数仅可选择一个
<i>ENET_PTP_ADD_TO_TIME</i>	更新值加到系统时间
<i>ENET_PTP_SUBTRACT_FROM_TIME</i>	将系统时间减去更新值
输入参数{in}	
second	秒值，范围(0~0xFFFF FFFF)
输入参数{in}	
subsecond	亚秒值，精度为0.46 ns，范围(0~0x7FFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize system time with timestamp update value */
enet_ptp_timestamp_update_config(ENET_PTP_ADD_TO_TIME, 0, 0);
```

函数 enet_ptp_expected_time_config

函数enet_ptp_expected_time_config描述见下表:

表 3-286. 函数 enet_ptp_expected_time_config

函数名称	enet_ptp_expected_time_config
函数原型	void enet_ptp_expected_time_config(uint32_t second, uint32_t nanosecond);
功能描述	配置PTP期望时间
先决条件	-
被调用函数	-
输入参数{in}	
second	目标时间秒值, 范围(0~0xFFFF FFFF)
输入参数{in}	
nanosecond	目标时间纳秒值, 范围(0~0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the expected target time */
enet_ptp_expected_time_config(2000, 0);
```

函数 enet_ptp_system_time_get

函数enet_ptp_system_time_get描述见下表:

表 3-287. 函数 enet_ptp_system_time_get

函数名称	enet_ptp_system_time_get
函数原型	void enet_ptp_system_time_get(enet_ptp_systime_struct *systime_struct);
功能描述	获取PTP当前系统时间
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

systime_struct	PTP系统时间结构体指针，结构体成员介绍请参考 结构体 <i>enet_ptp_systime_struct</i>
返回值	
-	-

例如：

```
/* get the current system time */
enet_ptp_systime_struct systime;
enet_ptp_system_time_get(&systime);
```

函数 **enet_ptp_pps_output_frequency_config**

函数**enet_ptp_pps_output_frequency_config**描述见下表：

表 3-288. 函数 **enet_ptp_pps_output_frequency_config**

函数名称	enet_ptp_pps_output_frequency_config
函数原型	void enet_ptp_pps_output_frequency_config(uint32_t freq);
功能描述	配置PPS输出频率
先决条件	-
被调用函数	-
输入参数{in}	
freq	输出频率
<i>ENET_PPSOFC_1HZ</i>	PPS输出1Hz
<i>ENET_PPSOFC_2HZ</i>	PPS输出2Hz
<i>ENET_PPSOFC_4HZ</i>	PPS输出4Hz
<i>ENET_PPSOFC_8HZ</i>	PPS输出8Hz
<i>ENET_PPSOFC_16HZ</i>	PPS输出16Hz
<i>ENET_PPSOFC_32HZ</i>	PPS输出32Hz

<i>ENET_PPSOFC_64HZ</i>	PPS输出64Hz
<i>ENET_PPSOFC_128HZ</i>	PPS输出128Hz
<i>ENET_PPSOFC_256HZ</i>	PPS输出256Hz
<i>ENET_PPSOFC_512HZ</i>	PPS输出512Hz
<i>ENET_PPSOFC_1024HZ</i>	PPS输出1024Hz
<i>ENET_PPSOFC_2048HZ</i>	PPS输出2048Hz
<i>ENET_PPSOFC_4096HZ</i>	PPS输出4096Hz
<i>ENET_PPSOFC_8192HZ</i>	PPS输出8192Hz
<i>ENET_PPSOFC_16384HZ</i>	PPS输出16384Hz
<i>ENET_PPSOFC_32768HZ</i>	PPS输出32768Hz
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PPS output frequency as 1Hz */
enet_ptp_pps_output_frequency_config(ENET_PPSOFC_1HZ);
```

函数 **enet_ptp_start**

函数The **enet_ptp_start**描述见下表：

表 3-289. 函数 **enet_ptp_start**

函数名称	enet_ptp_start
------	----------------

函数原型	<code>void enet_ptp_start(int32_t updatemethod, uint32_t init_sec, uint32_t init_subsec, uint32_t carry_cfg, uint32_t accuracy_cfg);</code>
功能描述	配置和启动PTP时间戳计数器
先决条件	-
被调用函数	-
输入参数{in}	
updatemethod	更新方式
<i>ENET_PTP_FINEMODE E</i>	精细校正
<i>ENET_PTP_COARSE MODE</i>	粗校正
输入参数{in}	
init_sec	初始化系统时间秒值
输入参数{in}	
init_subsec	初始化系统时间亚秒值
输入参数{in}	
carry_cfg	增加到时间戳加数寄存器的值（使用精细校正）
输入参数{in}	
accuracy_cfg	系统时间增加的亚秒值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* gconfigure and start PTP timestamp counter*/
enet_ptp_start(ENET_PTP_FINEMODE, 10, 10, 10, 10);
```

函数 **enet_ptp_finecorrection_adjfreq**

函数 `enet_ptp_finecorrection_adjfreq` 描述见下表：

表 3-290. 函数 `enet_ptp_finecorrection_adjfreq`

函数名称	enet_ptp_finecorrection_adjfreq
函数原型	void enet_ptp_finecorrection_adjfreq(int32_t carry_cfg);
功能描述	通过PTP时间戳加数寄存器精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
carry_cfg	增加到PTP加数寄存器的数值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* adjust frequency in fine method by configure addend register */
enet_ptp_finecorrection_adjfreq(10);
```

函数 enet_ptp_coarsecorrection_systime_update

函数enet_ptp_coarsecorrection_systime_update描述见下表：

表 3-291. 函数enet_ptp_coarsecorrection_systime_update

函数名称	enet_ptp_coarsecorrection_systime_update
函数原型	void enet_ptp_coarsecorrection_systime_update(enet_ptp_systime_struct *systime_struct);
功能描述	粗调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
systime_struct	t初始化结构体，结构体成员参考 表 3-199. 结构体 enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* update system time in coarse method */

enet_ptp_systime_struct systime_struct;

enet_ptp_coarsecorrection_systime_update (&systime_struct);
```

函数 **enet_ptp_finecorrection_settime**

函数enet_ptp_finecorrection_settime描述见下表：

表 3-292. 函数**enet_ptp_finecorrection_settime**

函数名称	enet_ptp_finecorrection_settime
函数原形	void enet_ptp_finecorrection_settime(enet_ptp_systime_struct * systime_struct);
功能描述	精调系统时间
先决条件	-
被调用函数	-
输入参数{in}	
systime_struct	初始化结构体，结构体成员参考 表 3-199. 结构体 enet_ptp_systime_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set system time in fine method */

enet_ptp_systime_struct systime_struct;

enet_ptp_finecorrection_settime (&systime_struct);
```

函数 **enet_ptp_flag_get**

函数enet_ptp_flag_get描述见下表：

表 3-293. 函数**enet_ptp_flag_get**

函数名称	enet_ptp_flag_get
------	-------------------

函数原形	FlagStatus enet_ptp_flag_get(uint32_t flag);
功能描述	获取PTP标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要检查的PTP标志位
ENET_PTP_ADDEND_UPDATE	加数寄存器更新
ENET_PTP_SYSTIME_UPDATE	时间戳更新
ENET_PTP_SYSTIME_INIT	时间戳初始化
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get the ptp flag status */
FlagStatus status = enet_ptp_flag_get(ENET_PTP_ADDEND_UPDATE);
```

函数 enet_initpara_reset

函数enet_initpara_reset描述见下表：

表 3-294. 函数 enet_initpara_reset

函数名称	enet_initpara_reset
函数原型	void enet_initpara_reset(void);
功能描述	复位 ENET initpara struct, 需在enet_initpara_config()函数前调用
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* reset the ENET initpara struct */
enet_initpara_reset();
```

3.11. EXMC

外部存储器控制器EXMC，用来访问各种片外存储器。章节[3.11.1](#)描述了EXMC的寄存器列表，章节[0](#)对EXMC库函数进行说明。

3.11.1. 外设寄存器说明

EXMC寄存器列表如下表所示：

表 3-295. EXMC 寄存器

寄存器名称	寄存器描述
EXMC_SNCTL	SRAM/NOR Flash控制寄存器
EXMC_SNTCFG	SRAM/NOR Flash时序寄存器
EXMC_SNWTCFG	SRAM/NOR Flash写时序寄存器
EXMC_NPCTL	NAND flash/PC card控制寄存器
EXMC_NPINTEN	NAND flash/PC card中断使能寄存器
EXMC_NPCTCFG	NAND flash/PC card通用空间时序寄存器
EXMC_NPATCFG	NAND flash/PC card属性空间时序寄存器
EXMC_PIOTCFG3	PC card I/O空间时序寄存器
EXMC_NECC	NAND flash ECC结果寄存器

3.11.2. 外设库函数说明

EXMC库函数列表如下表所示：

表 3-296. EXMC 库函数

库函数名称	库函数描述
exmc_norsram_deinit	复位EXMC NOR/SRAM regionx
exmc_norsram_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_norsram_init	初始化EXMC NOR/SRAM regionx
exmc_norsram_enable	使能EXMC NOR/PSRAM regionx
exmc_norsram_disable	禁用EXMC NOR/PSRAM regionx
exmc_nand_deinit	复位EXMC NAND bankx
exmc_nand_struct_para_init	初始化结构体exmc_norsram_parameter_struct
exmc_nand_init	初始化EXMC NAND bankx
exmc_nand_enable	使能EXMC NAND bankx
exmc_nand_disable	禁用EXMC NAND bankx
exmc_pccard_deinit	复位EXMC PC card bank
exmc_pccard_struct_para_init	初始化结构体exmc_pccard_parameter_struct
exmc_pccard_init	初始化EXMC PC card bank
exmc_pccard_enable	使能EXMC PC card bank
exmc_pccard_disable	禁用EXMC PC card bank
exmc_norsram_page_size_config	配置CRAM页大小
exmc_nand_ecc_config	配置EXMC NAND ECC功能
exmc_ecc_get	获取EXMC ECC值
exmc_interrupt_enable	使能EXMC中断
exmc_interrupt_disable	禁用EXMC中断
exmc_flag_get	获取EXMC状态
exmc_flag_clear	清除EXMC状态
exmc_interrupt_flag_get	获取EXMC中断状态
exmc_interrupt_flag_clear	清除EXMC中断状态

结构体 exmc_norsram_timing_parameter_struct
表 3-297. 结构体 exmc_norsram_timing_parameter_struct

成员名称	功能描述
asyn_access_mode	异步访问模式
syn_data_latency	数据延迟
syn_clk_division	同步时钟分频比
bus_latency	总线延迟
asyn_data_setuptime	数据建立时间
asyn_address_holdtime	地址保持时间
asyn_address_setuptime	地址建立时间

结构体 exmc_norsram_parameter_struct
表 3-298. 结构体 exmc_norsram_parameter_struct

成员名称	功能描述
norsram_region	选择EXMC NOR/SRAM Region
write_mode	写模式（同步模式或者异步模式）
extended_mode	使能或者禁用扩展模式
asyn_wait	使能或者禁用异步等待功能
nwait_signal	在同步突发模式中，使能或者禁用NWAIT信号
memory_write	使能或者禁用写操作
nwait_config	配置NWAIT信号
wrap_burst_mode	使能或者禁用非对齐成组模式
nwait_polarity	指定NWAIT的极性
burst_mode	使能或者禁用突发模式
databus_width	指定外部存储器数据总线宽度
memory_type	指定外部存储器的类型
address_data_mux	数据线/地址线复用是否复用

成员名称	功能描述
read_write_timing	未用扩展模式时，读时序参数和写时序参数；或采用扩展模式时，读时序参数
write_timing	使用扩展模式时，写时序参数

结构体 exmc_nand_pccard_timing_parameter_struct

表 3-299. 结构体 exmc_nand_pccard_timing_parameter_struct

成员名称	功能描述
databus_hiztime	写操作时数据总线高阻时间
holdtime	地址保持时间（写操作时数据保持时间）
waittime	等待时间（保持命令的最长时间）
setuptime	地址信号的建立时间

结构体 exmc_nand_parameter_struct

表 3-300. 结构体 exmc_nand_parameter_struct

成员名称	功能描述
nand_bank	选择EXMC NAND Bank
ecc_size	ECC块大小
atr_latency	ALE至RE的延迟
ctr_latency	CLE至RE的延迟
ecc_logic	配置ECC使能或禁用
databus_width	NAND flash数据宽度
wait_feature	配置NWAIT信号使能或禁用
common_space_timing	NAND flash通用空间时序配置
attribute_space_timing	NAND flash属性空间时序配置

结构体 exmc_pccard_parameter_struct

表 3-301. 结构体 exmc_pccard_parameter_struct

成员名称	功能描述
atr_latency	ALE至RE的延迟

成员名称	功能描述
ctr_latency	CLE至RE的延迟
wait_feature	配置NWAIT信号使能或禁用
common_space_timing	PC card通用空间时序配置
attribute_space_timing	PC card属性空间时序配置
io_space_timing	PC card I/O空间时序配置

函数 **exmc_norsram_deinit**

函数exmc_norsram_deinit描述见下表：

表 3-302. 函数 **exmc_norsram_deinit**

函数名称	exmc_norsram_deinit
函数原型	void exmc_norsram_deinit(uint32_t exmc_norsram_region);
功能描述	复位NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_deinit(EXMC_BANK0_NORSRAM_REGION1);
```

函数 exmc_norsram_struct_para_init

函数exmc_norsram_struct_para_init描述见下表:

表 3-303. 函数 exmc_norsram_struct_para_init

函数名称	exmc_norsram_struct_para_init
函数原型	void exmc_norsram_struct_para_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化结构体exmc_norsram_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_init_struct	初始化结构体, 结构体成员参考 表 3-298. 结构体 exmc_norsram_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize the struct nor_init_struct */

exmc_norsram_parameter_struct nor_init_struct;
exmc_norsram_struct_para_init (&nor_init_struct);
```

函数 exmc_norsram_init

函数exmc_norsram_init描述见下表:

表 3-304. 函数 exmc_norsram_init

函数名称	exmc_norsram_init
函数原型	void exmc_norsram_init(exmc_norsram_parameter_struct* exmc_norsram_init_struct);
功能描述	初始化NOR/SRAM region
先决条件	-
被调用函数	-

输入参数{in}	
exmc_norsram_ini_t_struct	初始化结构体, 结构体成员参考 表 3-298. 结构体 exmc_norsram_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* initialize EXMC NOR/SRAM bank */

exmc_norsram_parameter_struct lcd_init_struct;

exmc_norsram_timing_parameter_struct lcd_timing_init_struct;

/* configure timing parameter */

lcd_timing_init_struct.asyn_access_mode = EXMC_ACCESS_MODE_A;

lcd_timing_init_struct.syn_data_latency = EXMC_DATALAT_2_CLK;

lcd_timing_init_struct.syn_clk_division = EXMC_SYN_CLOCK_RATIO_DISABLE;

lcd_timing_init_struct.bus_latency = 1;

lcd_timing_init_struct.asyn_data_setuptime = 5;

lcd_timing_init_struct.asyn_address_holdtime = 2;

lcd_timing_init_struct.asyn_address_setuptime = 2;

/* configure EXMC bus parameters */

lcd_init_struct.norsram_region = EXMC_BANK0_NORSRAM_REGION1;

lcd_init_struct.write_mode = EXMC_ASYN_WRITE;

lcd_init_struct.extended_mode = DISABLE;

lcd_init_struct.asyn_wait = DISABLE;

lcd_init_struct.nwait_signal = DISABLE;

lcd_init_struct.memory_write = ENABLE;

lcd_init_struct.nwait_config = EXMC_NWAIT_CONFIG_BEFORE;

lcd_init_struct.wrap_burst_mode = DISABLE;

lcd_init_struct.nwait_polarity = EXMC_NWAIT_POLARITY_LOW;

```

```

lcd_init_struct.burst_mode = DISABLE;
lcd_init_struct.databus_width = EXMC_NOR_DATABUS_WIDTH_16B;
lcd_init_struct.memory_type = EXMC_MEMORY_TYPE_SRAM;
lcd_init_struct.address_data_mux = DISABLE;
lcd_init_struct.read_write_timing = &lcd_timing_init_struct;
lcd_init_struct.write_timing = &lcd_timing_init_struct;
exmc_norsram_init(&lcd_init_struct);

```

函数 exmc_norsram_enable

函数exmc_norsram_enable描述见下表:

表 3-305. 函数 exmc_norsram_enable

函数名称	exmc_norsram_enable
函数原型	void exmc_norsram_enable(uint32_t exmc_norsram_region);
功能描述	使能EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_re gion	EXMC NOR/SRAM region
EXMC_BANK0_NO RSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_enable(EXMC_BANK0_NORSRAM_REGION1);

```

函数 exmc_norsram_disable

函数exmc_norsram_disable描述见下表:

表 3-306. 函数 exmc_norsram_disable

函数名称	exmc_norsram_disable
函数原型	void exmc_norsram_disable(uint32_t exmc_norsram_region);
功能描述	禁用EXMC NOR/SRAM region
先决条件	-
被调用函数	-
输入参数{in}	
exmc_norsram_region	EXMC NOR/SRAM region
EXMC_BANK0_NORSRAM_REGIONx	x=0,1,2,3
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the EXMC NOR/SRAM region1 of bank0 */
exmc_norsram_disable(EXMC_BANK0_NORSRAM_REGION1);
```

函数 exmc_nand_deinit

函数exmc_nand_deinit描述见下表：

表 3-307. 函数 exmc_nand_deinit

函数名称	exmc_nand_deinit
函数原型	void exmc_nand_deinit(uint32_t exmc_nand_bank);
功能描述	复位EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NA	x=1,2

ND	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC NOR/SRAM bank1 */
exmc_norsram_deinit(EXMC_BANK1_NAND);
```

函数 exmc_nand_struct_para_init

函数exmc_nand_struct_para_init描述见下表：

表 3-308. 函数 exmc_nand_struct_para_init

函数名称	exmc_nand_struct_para_init
函数原型	void exmc_nand_struct_para_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化结构体exmc_nand_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_init_st ruct	初始化结构体，结构体成员参考 表 3-300. 结构体 exmc_nand_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct nand_init_struct */

exmc_nand_parameter_struct nand_init_struct;
exmc_nand_struct_para_init (&nand_init_struct);
```

函数 exmc_nand_init

函数exmc_nand_init描述见下表:

表 3-309. 函数 exmc_nand_init

函数名称	exmc_nand_init
函数原型	void exmc_nand_init(exmc_nand_parameter_struct* exmc_nand_init_struct);
功能描述	初始化EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_init_st ruct	初始化结构体, 结构体成员参考 表 3-300. 结构体 exmc_nand_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```

exmc_nand_parameter_struct nand_init_struct;
exmc_nand_pccard_timing_parameter_struct nand_timing_init_struct;

/* EXMC configuration */

nand_timing_init_struct.setuptime = 5;
nand_timing_init_struct.waittime = 4;
nand_timing_init_struct.holdtime = 2;
nand_timing_init_struct.databus_hiztime = 2;
nand_init_struct.nand_bank = EXMC_BANK1_NAND;
nand_init_struct.ecc_size = EXMC_ECC_SIZE_2048BYTES;
nand_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
nand_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
nand_init_struct.ecc_logic = ENABLE;
nand_init_struct.databus_width = EXMC_NAND_DATABUS_WIDTH_8B;

```

```

nand_init_struct.wait_feature = ENABLE;
nand_init_struct.common_space_timing = &nand_timing_init_struct;
nand_init_struct.attribute_space_timing = &nand_timing_init_struct;
exmc_nand_init(&nand_init_struct);

```

函数 exmc_nand_enable

函数exmc_nand_enable描述见下表:

表 3-310. 函数 exmc_nand_enable

函数名称	exmc_nand_enable
函数原型	void exmc_nand_enable(uint32_t exmc_nand_bank);
功能描述	使能EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NAND	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* enable EXMC NAND bank1 */
exmc_nand_enable(EXMC_BANK1_NAND);

```

函数 exmc_nand_disable

函数exmc_nand_disable描述见下表:

表 3-311. 函数 exmc_nand_disable

函数名称	exmc_nand_disable
函数原型	exmc_nand_disable(uint32_t exmc_nand_bank);

功能描述	禁用EXMC NAND bank
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NA ND	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXMC NAND bank1 */
exmc_nand_disable(EXMC_BANK1_NAND);
```

函数 exmc_pccard_deinit

函数exmc_pccard_deinit描述见下表：

表 3-312. 函数 exmc_pccard_deinit

函数名称	exmc_pccard_deinit
函数原型	void exmc_pccard_deinit(void);
功能描述	复位EXMC PC card bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* deinitialize EXMC PC card bank */
```

```
exmc_pccard_deinit();
```

函数 exmc_pccard_struct_para_init

函数exmc_pccard_struct_para_init描述见下表：

表 3-313. 函数 exmc_pccard_struct_para_init

函数名称	exmc_pccard_struct_para_init
函数原型	void exmc_pccard_struct_para_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);
功能描述	初始化结构体exmc_pccard_parameter_struct
先决条件	-
被调用函数	-
输入参数{in}	
exmc_pccard_init_struct	初始化结构体，结构体成员参考 表 3-301. 结构体 exmc_pccard_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the struct pccard_init_struct */

exmc_pccard_parameter_struct pccard_init_struct;
exmc_pccard_struct_para_init (&pccard_init_struct);
```

函数 exmc_pccard_init

函数exmc_pccard_init描述见下表：

表 3-314. 函数 exmc_pccard_init

函数名称	exmc_pccard_init
函数原型	void exmc_pccard_init(exmc_pccard_parameter_struct* exmc_pccard_init_struct);

功能描述	初始化EXMC PC card bank
先决条件	-
被调用函数	-
输入参数{in}	
<code>exmc_pccard_init_struct</code>	初始化结构体, 结构体成员参考 表 3-301. 结构体 exmc_pccard_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如：

```

exmc_pccard_parameter_struct pccard_init_struct;
exmc_nand_pccard_timing_parameter_struct pccard_timing_init_struct;
/* EXMC configuration */
pccard_timing_init_struct.setuptime = 5;
pccard_timing_init_struct.waittime = 4;
pccard_timing_init_struct.holdtime = 2;
pccard_timing_init_struct.databus_hiztime = 2;
pccard_init_struct.atr_latency = EXMC_ALE_RE_DELAY_1_HCLK;
pccard_init_struct.ctr_latency = EXMC_CLE_RE_DELAY_1_HCLK;
pccard_init_struct.wait_feature = ENABLE;
pccard_init_struct.common_space_timing = & pccard_timing_init_struct;
pccard_init_struct.attribute_space_timing = & pccard_timing_init_struct;
pccard_init_struct.io_space_timing = & pccard_timing_init_struct;
exmc_pccard_init(&pccard_init_struct);

```

函数 `exmc_pccard_enable`

函数`exmc_pccard_enable`描述见下表：

表 3-315. 函数 `exmc_pccard_enable`

函数名称	<code>exmc_pccard_enable</code>
------	---------------------------------

函数原型	void exmc_pccard_enable(void);
功能描述	使能EXMC PC card bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable EXMC PC card bank */

exmc_pccard_enable();
```

函数 **exmc_pccard_disable**

函数exmc_pccard_disable描述见下表：

表 3-316. 函数 **exmc_pccard_disable**

函数名称	exmc_pccard_disable
函数原型	void exmc_pccard_disable(void);
功能描述	禁用EXMC PC card bank
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXMC PC card bank */
```

```
exmc_pccard_disable();
```

函数 exmc_norsram_page_size_config

函数exmc_norsram_page_size_config描述见下表:

表 3-317. 函数 exmc_norsram_page_size_config

函数名称	exmc_norsram_page_size_config
函数原型	void exmc_norsram_page_size_config(uint32_t page_size);
功能描述	配置CRAM页大小
先决条件	-
被调用函数	-
输入参数{in}	
page_size	CRAM页大小
<i>EXMC_CRAM_AUT_O_SPLIT</i>	页边界自动突发分割
<i>EXMC_CRAM_PAGE_SIZE_128_BYTE</i>	页大小128字节
<i>EXMC_CRAM_PAGE_SIZE_256_BYTE</i>	页大小256字节
<i>EXMC_CRAM_PAGE_SIZE_512_BYTE</i>	页大小512字节
<i>EXMC_CRAM_PAGE_SIZE_1024_BYT_ES</i>	页大小1024字节
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure CRAM page size */
```

exmc_norsram_page_size_config(EXMC_CRAM_PAGE_SIZE_128_BYTES);

函数 exmc_nand_ecc_config

函数exmc_nand_ecc_config描述见下表:

表 3-318. 函数 exmc_nand_ecc_config

函数名称	exmc_nand_ecc_config
函数原型	void exmc_nand_ecc_config(uint32_t exmc_nand_bank, ControlStatus newvalue);
功能描述	使能或禁用EXMC NAND ECC功能
先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
EXMC_BANKx_NAND	x=1,2
输入参数{in}	
newvalue	ENABLE或DISABLE
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the EXMC NAND ECC function */

exmc_nand_ecc_config(EXMC_BANK1_NAND, ENABLE);
```

函数 exmc_ecc_get

函数exmc_ecc_get描述见下表:

表 3-319. 函数 exmc_ecc_get

函数名称	exmc_ecc_get
函数原型	uint32_t exmc_ecc_get(uint32_t exmc_nand_bank);
功能描述	获取EXMC NAND ECC值

先决条件	-
被调用函数	-
输入参数{in}	
exmc_nand_bank	EXMC NAND bank
<i>EXMC_BANKx_NA ND</i>	x=1,2
输出参数{out}	
-	-
返回值	
uint32_t	ECC计算的值

例如：

```
/* get the EXMC ECC value */

uint32_t ecc_value;

ecc_value = exmc_ecc_get(EXMC_BANK1_NAND);
```

函数 **exmc_interrupt_enable**

函数**exmc_interrupt_enable**描述见下表：

表 3-320. 函数 **exmc_interrupt_enable**

函数名称	exmc_interrupt_enable
函数原型	void exmc_interrupt_enable(uint32_t exmc_bank,uint32_t interrupt);
功能描述	使能EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC</i>	PC Card bank

CARD	
输入参数{in}	
interrupt	EXMC中断状态
<i>EXMC_NAND_PCCARD_INT_FLAG_RISE</i> <i>SE</i>	上升沿中断
<i>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</i>	高电平中断
<i>EXMC_NAND_PCCARD_INT_FLAG_FALL</i> <i>ALL</i>	下降沿中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable rising edge interrupt and flag t*/
exmc_interrupt_enable(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

函数 **exmc_interrupt_disable**

函数**exmc_interrupt_disable**描述见下表：

表 3-321. 函数 **exmc_interrupt_disable**

函数名称	exmc_interrupt_disable
函数原型	void exmc_interrupt_disable(uint32_t exmc_bank,uint32_t interrupt);
功能描述	禁用EXMC中断
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK1_NAND</i>	NAND bank1

<i>EXMC_BANK2_NAND</i>	NAND bank2
<i>EXMC_BANK3_PCCARD</i>	PC Card bank
输入参数{in}	
interrupt	EXMC中断状态
<i>EXMC_NAND_PCCARD_INT_FLAG_RISE</i>	上升沿中断
<i>EXMC_NAND_PCCARD_INT_FLAG_LEVEL</i>	高电平中断
<i>EXMC_NAND_PCCARD_INT_FLAG_FALL</i>	下降沿中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable rising edge interrupt and flag */

exmc_interrupt_disable(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

函数 exmc_flag_get

函数exmc_flag_get描述见下表：

表 3-322. 函数 exmc_flag_get

函数名称	exmc_flag_get
函数原型	FlagStatus exmc_flag_get(uint32_t exmc_bank,uint32_t flag);
功能描述	获取EXMC状态
先决条件	-
被调用函数	-

输入参数{in}	
exmc_bank	EXMC外设
EXMC_BANK1_NA ND	NAND bank1
EXMC_BANK2_NA ND	NAND bank2
EXMC_BANK3_PC CARD	PC Card bank
输入参数{in}	
flag	EXMC标志状态
EXMC_NAND_PCC ARD_FLAG_RISE	上升沿状态
EXMC_NAND_PCC ARD_FLAG_LEVEL	高电平状态
EXMC_NAND_PCC ARD_FLAG_FALL	下降沿状态
EXMC_NAND_PCC ARD_FLAG_FIFOE	FIFO空状态
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check rising edge status is set or not*/
if(RESET != exmc_flag_get(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_FLAG_RISE));
```

函数 **exmc_flag_clear**

函数**exmc_flag_clear**描述见下表：

表 3-323. 函数 **exmc_flag_clear**

函数名称	exmc_flag_clear
函数原型	FlagStatus exmc_flag_clear (uint32_t exmc_bank,uint32_t flag);

功能描述	清除EXMC状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC外设
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
输入参数{in}	
flag	EXMC标志状态
<i>EXMC_NAND_PCC ARD_FLAG_RISE</i>	上升沿状态
<i>EXMC_NAND_PCC ARD_FLAG_LEVEL</i>	高电平状态
<i>EXMC_NAND_PCC ARD_FLAG_FALL</i>	下降沿状态
<i>EXMC_NAND_PCC ARD_FLAG_FIFOE</i>	FIFO空状态
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear rising edge status */
exmc_flag_clear(EXMC_BANK1_NAND, EXMC_NAND_PCCARD_FLAG_RISE);
```

函数 exmc_interrupt_flag_get

函数exmc_interrupt_flag_get描述见下表：

表 3-324. 函数 exmc_interrupt_flag_get

函数名称	exmc_interrupt_flag_get
函数原型	FlagStatus exmc_interrupt_flag_get(uint32_t exmc_bank,uint32_t interrupt);
功能描述	获取EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC中断状态
EXMC_BANK1_NA ND	NAND bank1
EXMC_BANK2_NA ND	NAND bank2
EXMC_BANK3_PC CARD	PC Card bank
输入参数{in}	
interrupt	中断
EXMC_NAND_PCC ARD_INT_FLAG_RI SE	上升沿中断
EXMC_NAND_PCC ARD_INT_FLAG_L EVEL	高电平中断
EXMC_NAND_PCC ARD_INT_FLAG_F ALL	下降沿中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* check rising edge interrupt and flag is set or not*/
```

```
if(RESET != exmc_interrupt_flag_get(EXMC_BANK1_NAND,
```

EXMC_NAND_PCCARD_INT_FLAG_RISE));

函数 exmc_interrupt_flag_clear

函数exmc_interrupt_flag_clear描述见下表：

表 3-325. 函数 exmc_interrupt_flag_clear

函数名称	exmc_interrupt_flag_clear
函数原型	void exmc_interrupt_flag_clear(uint32_t exmc_bank,uint32_t interrupt);
功能描述	清除EXMC中断状态
先决条件	-
被调用函数	-
输入参数{in}	
exmc_bank	EXMC中断状态
<i>EXMC_BANK1_NA ND</i>	NAND bank1
<i>EXMC_BANK2_NA ND</i>	NAND bank2
<i>EXMC_BANK3_PC CARD</i>	PC Card bank
输入参数{in}	
interrupt	中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_RI SE</i>	上升沿中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_L EVEL</i>	高电平中断
<i>EXMC_NAND_PCC ARD_INT_FLAG_F ALL</i>	下降沿中断
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* clearrising edge interrupt and flag */

exmc_interrupt_flag_clear(EXMC_BANK1_NAND,
EXMC_NAND_PCCARD_INT_FLAG_RISE);
```

3.12. EXTI

EXTI是MCU中的中断/事件控制器，包括20个相互独立的边沿检测电路并且能够向处理器内核产生中断请求或唤醒事件。章节[3.12.1](#)描述了EXTI的寄存器列表，章节[3.12.2](#)对EXTI库函数进行说明。

3.12.1. 外设寄存器说明

EXTI寄存器列表如下表所示：

表 3-326. EXTI 寄存器

寄存器名称	寄存器描述
EXTI_INTEN	中断使能寄存器
EXTI_EVEN	事件使能寄存器
EXTI_RTEN	上升沿触发使能寄存器
EXTI_FTEN	下降沿触发使能寄存器
EXTI_SWIEV	软件中断事件寄存器
EXTI_PD	挂起寄存器

3.12.2. 外设库函数说明

EXTI库函数列表如下表所示：

表 3-327. EXTI 库函数

库函数名称	库函数描述
exti_deinit	复位EXTI
exti_init	初始化EXTI线x
exti_interrupt_enable	EXTI线x中断使能

库函数名称	库函数描述
exti_interrupt_disable	EXTI线x中断禁能
exti_event_enable	EXTI线x事件使能
exti_event_disable	EXTI线x事件禁能
exti_software_interrupt_enable	EXTI线x软件中断使能
exti_software_interrupt_disable	EXTI线x软件中断禁能
exti_flag_get	获取EXTI线x标志位
exti_flag_clear	清除EXTI线x标志位
exti_interrupt_flag_get	获取EXTI线x中断标志位
exti_interrupt_flag_clear	清除EXTI线x中断标志位

函数 exti_deinit

函数exti_deinit描述见下表:

表 3-328. 函数 exti_deinit

函数名称	exti_deinit
函数原形	void exti_deinit(void);
功能描述	复位EXTI，将EXTI的所有寄存器恢复成初始值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize the EXTI */
exti_deinit();
```

函数 exti_init

函数exti_init描述见下表：

表 3-329. 函数 exti_init

函数名称	exti_init
函数原形	void exti_init(exti_line_enum linex, exti_mode_enum mode, exti_trig_type_enum trig_type);
功能描述	初始化EXTI线x
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输入参数{in}	
mode	EXTI模式
EXTI_INTERRUPT	中断模式
EXTI_EVENT	事件模式
输入参数{in}	
trig_type	触发类型
EXTI_TRIG_RISING	上升沿触发
EXTI_TRIG_FALLING	下降沿触发
EXTI_TRIG_BOTH	上升沿和下降沿均触发
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure EXTI_0 */
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_BOTH);
```

函数 exti_interrupt_enable

函数exti_interrupt_enable描述见下表:

表 3-330. 函数 exti_interrupt_enable

函数名称	exti_interrupt_enable
函数原形	void exti_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the interrupts from EXTI line 0 */
exti_interrupt_enable(EXTI_0);
```

函数 exti_interrupt_disable

函数exti_interrupt_disable描述见下表:

表 3-331. 函数 exti_interrupt_disable

函数名称	exti_interrupt_disable
函数原形	void exti_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x

<i>EXTI_x</i>	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the interrupts from EXTI line 0 */

exti_interrupt_disable(EXTI_0);
```

函数 exti_event_enable

函数exti_event_enable描述见下表：

表 3-332. 函数 exti_event_enable

函数名称	exti_event_enable
函数原形	void exti_event_enable(exti_line_enum linex);
功能描述	EXTI线x事件使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
<i>EXTI_x</i>	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the events from EXTI line 0 */

exti_event_enable(EXTI_0);
```

函数 exti_event_disable

函数exti_event_disable描述见下表：

表 3-333. 函数 exti_event_disable

函数名称	exti_event_disable
函数原形	void exti_event_disable(exti_line_enum linex);
功能描述	EXTI线x事件禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the events from EXTI line 0 */
exti_event_disable(EXTI_0);
```

函数 exti_software_interrupt_enable

函数exti_software_interrupt_enable描述见下表：

表 3-334. 函数 exti_software_interrupt_enable

函数名称	exti_software_interrupt_enable
函数原形	void exti_software_interrupt_enable(exti_line_enum linex);
功能描述	EXTI线x软件中断使能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable EXTI line 0 software interrupt */
exti_software_interrupt_enable(EXTI_0);
```

函数 exti_software_interrupt_disable

函数exti_software_interrupt_disable描述见下表：

表 3-335. 函数 exti_software_interrupt_disable

函数名称	exti_software_interrupt_disable
函数原形	void exti_software_interrupt_disable(exti_line_enum linex);
功能描述	EXTI线x软件中断禁能
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable EXTI line 0 software interrupt */
exti_software_interrupt_disable(EXTI_0);
```

函数 exti_flag_get

函数exti_flag_get描述见下表：

表 3-336. 函数 exti_flag_get

函数名称	exti_flag_get
------	---------------

函数原形	FlagStatus exti_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get EXTI line 0 flag status */
FlagStatus state = exti_flag_get(EXTI_0);
```

函数 exti_flag_clear

函数exti_flag_clear描述见下表：

表 3-337. 函数 exti_flag_clear

函数名称	exti_flag_clear
函数原形	void exti_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* clear EXTI line 0 flag status */

exti_flag_clear(EXTI_0);
```

函数 exti_interrupt_flag_get

函数exti_interrupt_flag_get描述见下表：

表 3-338. 函数 exti_interrupt_flag_get

函数名称	exti_interrupt_flag_get
函数原形	FlagStatus exti_interrupt_flag_get(exti_line_enum linex);
功能描述	获取EXTI线x中断标志位
先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get EXTI line 0 interrupt flag status */

FlagStatus state = exti_interrupt_flag_get(EXTI_0);
```

函数 exti_interrupt_flag_clear

函数exti_interrupt_flag_clear描述见下表：

表 3-339. 函数 exti_interrupt_flag_clear

函数名称	exti_interrupt_flag_clear
函数原形	void exti_interrupt_flag_clear(exti_line_enum linex);
功能描述	清除EXTI线x中断标志位

先决条件	-
被调用函数	-
输入参数{in}	
linex	EXTI线x
EXTI_x	x=0,1,2..19
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear EXTI line 0 interrupt flag status */
exti_interrupt_flag_clear(EXTI_0);
```

3.13. FMC

FMC是MCU中的Flash控制器，其中包括存储数据的主编程块和选项字节。章节[3.13.1](#)描述了FMC的寄存器列表，章节[3.13.2](#)对FMC库函数进行说明。

3.13.1. 外设寄存器说明

FMC寄存器列表如下：

表 3-340. FMC 寄存器

寄存器	描述
FMC_WS	等待状态寄存器
FMC_KEY0	解锁寄存器0
FMC_OBKEY	选项字节解锁寄存器
FMC_STAT0	状态寄存器0
FMC_CTL0	控制寄存器0
FMC_ADDR0	地址寄存器0
FMC_OBSTAT	选项字节状态寄存器
FMC_WP	擦写/编程保护寄存器

寄存器	描述
FMC_KEY1	解锁寄存器1
FMC_STAT1	状态寄存器1
FMC_CTL1	控制寄存器1
FMC_ADDR1	地址寄存器1
FMC_WSEN	等待状态使能寄存器
FMC_PID	产品ID寄存器

3.13.2. 外设库函数说明

FMC固件库函数列举如下表：

表 3-341. FMC 固件库函数

函数名称	函数描述
fmc_wscnt_set	设置FMC等待状态计数值
fmc_unlock	解锁FMC主编程块操作
fmc_bank0_unlock	解锁FMC BANK0块操作
fmc_bank1_unlock	解锁FMC BANK1块操作
fmc_lock	锁定FMC主编程块操作
fmc_bank0_lock	锁定FMC BANK0块操作
fmc_bank1_lock	锁定FMC BANK1块操作
fmc_page_erase	FMC 页擦除
fmc_mass_erase	FMC 全片擦除
fmc_bank0_erase	FMC BANK0擦除
fmc_bank1_erase	FMC BANK1擦除
fmc_word_program	在相应地址全字编程
fmc_halfword_program	在相应地址半字编程
fmc_word_reprogram	在相应地址全字编程，无需检查该处内存是否擦除
ob_unlock	解锁选项字节操作
ob_lock	锁定选项字节操作
ob_erase	擦除选项字节

函数名称	函数描述
ob_write_protection_enable	使能写保护
ob_security_protection_config	配置安全保护
ob_user_write	写用户选项字节
ob_data_program	写数据选项字节
ob_user_get	获取用户选项字节
ob_data_get	获取数据选项字节
ob_write_protection_get	获取写保护选项字节
ob_spc_get	获取安全保护选项字节
fmc_interrupt_enable	使能FMC中断
fmc_interrupt_disable	除能FMC中断
fmc_flag_get	检查标志位是否置位
fmc_flag_clear	清除FMC标志
fmc_interrupt_flag_get	获取FMC中断标志状态
fmc_interrupt_flag_clear	清除FMC中断标志状态
fmc_bank0_state_get	获取FMC BANK0状态
fmc_bank1_state_get	获取FMC BANK1状态
fmc_bank0_ready_wait	检查FMC BANK0是否准备好
fmc_bank1_ready_wait	检查FMC BANK1是否准备好

枚举 fmc_state_enum

表 3-342. fmc_state_enum

枚举名称	枚举描述
FMC_READY	操作完成
FMC_BUSY	操作进行中
FMC_PGERR	编程错误
FMC_WPERR	写保护错误
FMC_TOERR	超时错误

枚举 fmc_int_enum

表 3-343. fmc_int_enum

枚举名称	枚举描述
FMC_INT_BANK0-END	使能FMC BANK0编程完成中断
FMC_INT_BANK0-ERR	使能FMC BANK0错误中断
FMC_INT_BANK1-END	使能FMC BANK1编程完成中断
FMC_INT_BANK1-ERR	使能FMC BANK1错误中断

枚举 fmc_flag_enum

表 3-344. fmc_flag_enum

枚举名称	枚举描述
FMC_FLAG_BANK0_BUSY	FMC BANK0忙碌标志
FMC_FLAG_BANK0_PGERR	FMC BANK0操作错误标志
FMC_FLAG_BANK0_WPERR	FMC BANK0写保护错误标志
FMC_FLAG_BANK0_END	FMC BANK0操作完成标志
FMC_FLAG_OBER_R	FMC选项字节错误标志
FMC_FLAG_BANK1_BUSY	FMC BANK1忙碌标志
FMC_FLAG_BANK1_PGERR	FMC BANK1操作错误标志
FMC_FLAG_BANK1_WPERR	FMC BANK1写保护错误标志
FMC_FLAG_BANK1_END	FMC BANK1操作完成标志

枚举 fmc_interrupt_flag_enum
表 3-345. fmc_interrupt_flag_enum

枚举名称	枚举描述
FMC_INT_FLAG_BANK0_PGERR	FMC BANK0操作错误中断标志
FMC_INT_FLAG_BANK0_WPERR	FMC BANK0写保护错误中断标志
FMC_INT_FLAG_BANK0_END	FMC BANK0操作完成中断标志
FMC_INT_FLAG_BANK1_PGERR	FMC BANK1操作错误中断标志
FMC_INT_FLAG_BANK1_WPERR	FMC BANK1写保护错误中断标志
FMC_INT_FLAG_BANK1_END	FMC BANK1操作完成中断标志

函数 fmc_wscnt_set

函数fmc_wscnt_set描述见下表:

表 3-346. 函数 fmc_wscnt_set

函数名称	fmc_wscnt_set
函数原型	void fmc_wscnt_set(uint32_t wscnt);
功能描述	设置等待状态计数值
先决条件	-
被调用函数	-
输入参数{in}	
wscnt	等待状态计数值
WS_WSCNT_0	FMC 0个等待状态
WS_WSCNT_1	FMC 1个等待状态
WS_WSCNT_2	FMC 2个等待状态
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* set the wait state counter value */

fmc_wscnt_set (WS_WSCNT_1);
```

函数 fmc_unlock

函数fmc_unlock描述见下表：

表 3-347. 函数 fmc_unlock

函数名称	fmc_unlock
函数原型	void fmc_unlock(void);
功能描述	解锁Flash操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the main FMC operation */

fmc_unlock( );
```

函数 fmc_bank0_unlock

函数fmc_bank0_unlock描述见下表：

表 3-348. 函数 fmc_bank0_unlock

函数名称	fmc_bank0_unlock
函数原型	void fmc_bank0_unlock(void);
功能描述	解锁Flash BANK0块操作

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the main FMC BANK0 operation */

fmc_bank0_unlock();
```

函数 **fmc_bank1_unlock**

函数fmc_bank1_unlock描述见下表：

表 3-349. 函数 **fmc_bank1_unlock**

函数名称	fmc_bank1_unlock
函数原型	void fmc_bank1_unlock(void);
功能描述	解锁Flash BANK1块操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the main FMC BANK1 operation */

fmc_bank1_unlock();
```

函数 fmc_lock

函数fmc_lock描述见下表:

表 3-350. 函数 fmc_lock

函数名称	fmc_lock
函数原型	void fmc_lock(void);
功能描述	锁定Flash操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* lock the main FMC operation */

fmc_lock( );
```

函数 fmc_bank0_lock

函数fmc_bank0_lock描述见下表:

表 3-351. 函数 fmc_bank0_lock

函数名称	fmc_bank0_lock
函数原型	void fmc_bank0_lock(void);
功能描述	锁定Flash BANK0操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* lock the main FMC BANK0 operation */

fmc_bank0_lock();
```

函数 fmc_bank1_lock

函数fmc_bank1_lock描述见下表：

表 3-352. 函数 fmc_bank1_lock

函数名称	fmc_bank1_lock
函数原型	void fmc_bank1_lock(void);
功能描述	锁定Flash BANK1操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the main FMC BANK1 operation */

fmc_bank1_lock();
```

函数 fmc_page_erase

函数fmc_page_erase描述见下表：

表 3-353. 函数 fmc_page_erase

函数名称	fmc_page_erase
函数原型	fmc_state_enum fmc_page_erase(uint32_t page_address);

功能描述	页擦除
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
输入参数{in}	
page_address	页擦除首地址
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* erase page */
fmc_unlock();
fmc_state_enum state = fmc_page_erase ( 0x08004000);
```

函数 fmc_mass_erase

函数fmc_mass_erase描述见下表:

表 3-354. 函数 fmc_mass_erase

函数名称	fmc_mass_erase
函数原型	fmc_state_enum fmc_mass_erase(void);
功能描述	全片擦除
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```

/* erase whole chip */

fmc_unlock();

fmc_state_enum state = fmc_mass_erase();

```

函数 **fmc_bank0_erase**

函数fmc_bank0_erase描述见下表：

表 3-355. 函数 fmc_bank0_erase

函数名称	fmc_bank0_erase
函数原型	fmc_state_enum fmc_bank0_erase(void);
功能描述	FMC BANK0擦除
先决条件	fmc_unlock/fmc_bank0_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如：

```

/* erase bank0 */

fmc_unlock();

fmc_state_enum state = fmc_bank0_erase();

```

函数 **fmc_bank1_erase**

函数fmc_bank1_erase描述见下表：

表 3-356. 函数 fmc_bank1_erase

函数名称	fmc_bank1_erase
函数原型	fmc_state_enum fmc_bank1_erase(void);
功能描述	FMC BANK1擦除
先决条件	fmc_unlock/fmc_bank1_unlock

被调用函数	fmc_bank1_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* erase bank1 */
fmc_unlock();
fmc_state_enum state = fmc_bank1_erase();
```

函数 fmc_word_program

函数fmc_word_program描述见下表:

表 3-357. 函数 fmc_word_program

函数名称	fmc_word_program
函数原型	fmc_state_enum fmc_word_program(uint32_t address, uint32_t data);
功能描述	对相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* program a word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_word_program (0x08004000, 0xaabbccdd);
```

函数 fmc_halfword_program

函数fmc_halfword_program描述见下表：

表 3-358. 函数 fmc_halfword_program

函数名称	fmc_halfword_program
函数原型	fmc_state_enum fmc_halfword_program(uint32_t address, uint16_t data);
功能描述	对相应地址半字编程
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如：

```
/* program a half word at the corresponding address */

fmc_unlock();

fmc_page_erase(0x08004000);

fmc_state_enum state = fmc_halfword_program (0x08004000, 0xaabb);
```

函数 fmc_word_reprogram

函数fmc_word_reprogram描述见下表：

表 3-359. 函数 fmc_word_reprogram

函数名称	fmc_word_reprogram
函数原型	fmc_state_enum fmc_word_reprogram(uint32_t address, uint32_t data);
功能描述	对相应地址全字编程
先决条件	fmc_unlock
被调用函数	fmc_bank0_ready_wait/ fmc_bank1_ready_wait
输入参数{in}	
address	编程地址
输入参数{in}	
data	编程数据
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* program a word at the corresponding address without erasing*/
fmc_unlock();
fmc_state_enum state = fmc_word_reprogram (0x08004000, 0xaabbccdd);
```

函数 ob_unlock

函数ob_unlock描述见下表:

表 3-360. 函数 ob_unlock

函数名称	ob_unlock
函数原型	void ob_unlock(void);
功能描述	解锁选项字节
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* unlock the option byte operation */

fmc_unlock( );
ob_unlock ( );
```

函数 ob_lock

函数ob_lock描述见下表：

表 3-361. 函数 ob_lock

函数名称	ob_lock
函数原型	void ob_lock(void);
功能描述	锁定选项字节操作
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* lock the option byte operation */

fmc_unlock( );
ob_lock ( );
```

函数 ob_erase

函数ob_erase描述见下表：

表 3-362. 函数 ob_erase

函数名称	ob_erase
函数原型	fmc_state_enum ob_erase(void);
功能描述	擦除选项字节
先决条件	ob_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* erase the FMC option byte */

fmc_unlock( );
ob_unlock( );
fmc_state enum state = ob_erase ( );
```

函数 ob_write_protection_enable

函数ob_write_protection_enable描述见下表:

表 3-363. 函数 ob_write_protection_enable

函数名称	ob_write_protection_enable
函数原型	fmc_state_enum ob_write_protection_enable(uint32_t ob_wp);
功能描述	使能写保护
先决条件	ob_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
ob_wp	写保护单元
OB_WP _x	特定写保护单元 (x= 0 ...31)

<code>OB_WP_ALL</code>	全片写保护
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* enable write protection */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_write_protection_enable (OB_WP7);
```

函数 `ob_security_protection_config`

函数`ob_security_protection_config`描述见下表:

表 3-364. 函数 `ob_security_protection_config`

函数名称	<code>ob_security_protection_config</code>
函数原型	<code>fmc_state_enum ob_security_protection_config (uint8_t ob_spc);</code>
功能描述	配置安全保护
先决条件	<code>ob_unlock</code>
被调用函数	<code>fmc_bank0_ready_wait</code>
输入参数{in}	
<code>ob_spc</code>	安全保护
<code>FMC_NSPC</code>	无安全保护
<code>FMC_USPC</code>	安全保护
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* enable security protection */
```

```

ob_unlock( );
fmc_state_enum state = ob_security_protection_config(FMC_USPC);

```

函数 **ob_user_write**

函数**ob_user_write**描述见下表：

表 3-365. 函数 **ob_user_write**

函数名称	ob_user_write
函数原型	fmc_state_enum ob_user_write(uint8_t ob_fwdgt, uint8_t ob_deepsleep, uint8_t ob_stby, uint8_t ob_boot);
功能描述	编辑用户选项字节
先决条件	ob_unlock
被调用函数	fmc_bank0_ready_wait
输入参数{in}	
ob_fwdgt	选项字节看门狗数值
OB_FWDGT_SOFT_WARE	软件看门狗
OB_FWDGT_HARD_WARE	硬件看门狗
输入参数{in}	
ob_deepsleep	选项字节深度睡眠复位值
OB_DEEPSLEEP_NRST	进入深度睡眠时不复位
OB_DEEPSLEEP_RST	进入深度睡眠时产生复位
输入参数{in}	
ob_stby	选项字节待机复位值
OB_STDBY_NRST	进入待机时不复位
OB_STDBY_RST	进入待机时产生复位
输入参数{in}	
ob_boot	选项字节待机复位值
OB_BOOT_BO	从BANK0启动

<code>OB_BOOT_B1</code>	从BANK1启动，若BANK1为空，则从BANK0启动
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，详细请参考 枚举fmc_state_enum

例如：

```
/* configure user option byte */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_user_write(OB_FWDGT_HW,OB_DEEPSLEEP_RST,
OB_STDBY_RST);
```

函数 `ob_data_program`

函数`ob_data_program`描述见下表：

表 3-366. 函数 `ob_data_program`

函数名称	<code>ob_data_program</code>
函数原型	<code>fmc_state_enum ob_data_program(uint32_t address, uint8_t data);</code>
功能描述	编程数据选项字节
先决条件	<code>ob_unlock</code>
被调用函数	<code>fmc_bank0_ready_wait</code>
输入参数{in}	
<code>address</code>	编程数据选项字节地址
输入参数{in}	
<code>data</code>	所编程数值
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，详细请参考 枚举fmc_state_enum

例如：

```

/* program option bytes data */

fmc_unlock( );

ob_unlock( );

fmc_state_enum state = ob_data_program (0xffff804, 0x56);

```

函数 **ob_user_get**

函数ob_user_get描述见下表：

表 3-367. 函数 ob_user_get

函数名称	ob_user_get
函数原型	uint8_t ob_user_get(void);
功能描述	获取用户选项字节
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	选项字节用户数值 (0xF0 – 0xFF)

例如：

```

/* get the FMC user option byte */

uint8_t user = ob_user_get ( );

```

函数 **ob_data_get**

函数ob_data_get描述见下表：

表 3-368. 函数 ob_data_get

函数名称	ob_data_get
函数原型	uint16_t ob_data_get(void);
功能描述	获取数据选项字节
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	选项字节数据值 (0x0 – 0xFF)

例如：

```
/* get the FMC data option byte */

uint16_t data = ob_data_get();
```

函数 **ob_write_protection_get**

函数ob_write_protection_get描述见下表：

表 3-369. 函数 **ob_write_protection_get**

函数名称	ob_write_protection_get
函数原型	uint32_t ob_write_protection_get(void);
功能描述	获取选项字节写保护数值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	选项字节写保护数值 (0x0 – 0xFFFFFFFF)

例如：

```
/* get the FMC option byte write protection */

uint32_t wp = ob_write_protection_get();
```

函数 ob_spc_get

函数ob_spc_get描述见下表:

表 3-370. 函数 ob_spc_get

函数名称	ob_spc_get
函数原型	FlagStatus ob_spc_get (void);
功能描述	获取安全保护状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如:

```
/* get the FMC option byte security protection */

FlagStatus spc = ob_spc_get();
```

函数 fmc_interrupt_enable

函数fmc_interrupt_enable描述见下表:

表 3-371. 函数 fmc_interrupt_enable

函数名称	fmc_interrupt_enable
函数原型	void fmc_interrupt_enable(uint32_t interrupt);
功能描述	使能FMC中断
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
interrupt	FMC中断
FMC_INT_BANK0-END	FMC BANK0编程完成中断

<i>FMC_INT_BANK0_ERR</i>	FMC BANK0错误中断
<i>FMC_INT_BANK1_END</i>	FMC BANK1编程完成中断
<i>FMC_INT_BANK1_ERR</i>	FMC BANK1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable FMC BANK0 end of program interrupt */

fmc_unlock( );
fmc_interrupt_enable(FMC_INT_BANK0_END);
```

函数 **fmc_interrupt_disable**

函数fmc_interrupt_disable描述见下表：

表 3-372. 函数 **fmc_interrupt_disable**

函数名称	fmc_interrupt_disable
函数原型	void fmc_interrupt_disable(uint32_t interrupt);
功能描述	除能FMC中断
先决条件	fmc_unlock
被调用函数	-
输入参数{in}	
interrupt	FMC中断
<i>FMC_INT_BANK0_END</i>	FMC BANK0编程完成中断
<i>FMC_INT_BANK0_ERR</i>	FMC BANK0错误中断
<i>FMC_INT_BANK1_END</i>	FMC BANK1编程完成中断

<i>FMC_INT_BANK1_ERR</i>	FMC BANK1错误中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable FMC BANK0 end of program interrupt */

fmc_unlock( );

fmc_interrupt_disable(FMC_INT_BANK0_END);
```

函数 fmc_flag_get

函数fmc_flag_get描述见下表：

表 3-373. 函数 fmc_flag_get

函数名称	fmc_flag_get
函数原型	FlagStatus fmc_flag_get(uint32_t flag);
功能描述	检查标志是否置位
先决条件	-
被调用函数	-
输入参数{in}	
flag	检查FMC标志
<i>FMC_FLAG_BANK0_BUSY</i>	FMC BANK0忙碌标志
<i>FMC_FLAG_BANK0_PGERR</i>	FMC BANK0操作错误标志
<i>FMC_FLAG_BANK0_WPERR</i>	FMC BANK0写保护错误标志
<i>FMC_FLAG_BANK0_END</i>	FMC BANK0操作完成标志
<i>FMC_FLAG_OBER_R</i>	FMC选项字节错误标志

<i>FMC_FLAG_BANK1_BUSY</i>	FMC BANK1忙碌标志
<i>FMC_FLAG_BANK1_PGEERR</i>	FMC BANK1操作错误标志
<i>FMC_FLAG_BANK1_WPERR</i>	FMC BANK1写保护错误标志
<i>FMC_FLAG_BANK1_END</i>	FMC BANK1操作完成标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get FMC bank0 end of operation flag bit */
FlagStatus flag = fmc_flag_get(FMC_FLAG_BANK0_END);
```

函数 **fmc_flag_clear**

函数fmc_flag_clear描述见下表：

表 3-374. 函数 **fmc_flag_clear**

函数名称	fmc_flag_clear
函数原型	void fmc_flag_clear(uint32_t flag);
功能描述	清除FMC标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC标志
<i>FMC_FLAG_BANK0_PGEERR</i>	FMC BANK0操作错误标志
<i>FMC_FLAG_BANK0_WPERR</i>	FMC BANK0写保护错误标志
<i>FMC_FLAG_BANK0_END</i>	FMC BANK0操作完成标志

<i>_END</i>	
<i>FMC_FLAG_BANK1</i> <i>_PGERR</i>	FMC BANK1操作错误标志
<i>FMC_FLAG_BANK1</i> <i>_WPERR</i>	FMC BANK1写保护错误标志
<i>FMC_FLAG_BANK1</i> <i>_END</i>	FMC BANK1操作完成标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear FMC bank0 operation error flag */
FlagStatus flag = fmc_flag_clear(FMC_FLAG_BANK0_PGERR);
```

函数 **fmc_interrupt_flag_get**

函数fmc_interrupt_flag_get描述见下表：

表 3-375. 函数 **fmc_interrupt_flag_get**

函数名称	fmc_interrupt_flag_get
函数原型	FlagStatus fmc_interrupt_flag_get(fmc_interrupt_flag_enum flag);
功能描述	获取FMC中断标志状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	中断标志
<i>FMC_INT_FLAG_BANK0_PGERR</i>	FMC BANK0操作错误中断标志
<i>FMC_INT_FLAG_BANK0_WPERR</i>	FMC BANK0写保护错误中断标志
<i>FMC_INT_FLAG_BANK0_END</i>	FMC BANK0操作完成中断标志

<i>FMC_INT_FLAG_B</i> <i>ANK0_PGERR</i>	FMC BANK1操作错误中断标志
<i>FMC_INT_FLAG_B</i> <i>ANK0_WPERR</i>	FMC BANK1写保护错误中断标志
<i>FMC_INT_FLAG_B</i> <i>ANK0_END</i>	FMC BANK1操作完成中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get FMC bank0 operation error interrupt flag bit */

FlagStatus flag = fmc_interrupt_flag_get (FMC_INT_FLAG_BANK0_PGERR);
```

函数 **fmc_interrupt_flag_clear**

函数fmc_interrupt_flag_clear描述见下表：

表 3-376. 函数 **fmc_interrupt_flag_clear**

函数名称	fmc_interrupt_flag_clear
函数原型	FlagStatus fmc_interrupt_flag_clear (fmc_interrupt_flag_enum flag);
功能描述	清除FMC中断标志
先决条件	-
被调用函数	-
输入参数{in}	
flag	清除FMC中断标志
<i>FMC_INT_FLAG_B</i> <i>ANK0_PGERR</i>	FMC BANK0操作错误中断标志
<i>FMC_INT_FLAG_B</i> <i>ANK0_WPERR</i>	FMC BANK0写保护错误中断标志
<i>FMC_INT_FLAG_B</i> <i>ANK0_END</i>	FMC BANK0操作完成中断标志
<i>FMC_INT_FLAG_B</i>	FMC BANK1操作错误中断标志

<code>ANK0_PGERR</code>	
<code>FMC_INT_FLAG_B</code> <code>ANK0_WPERR</code>	FMC BANK1写保护错误中断标志
<code>FMC_INT_FLAG_B</code> <code>ANK0_END</code>	FMC BANK1操作完成中断标志
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET or RESET

例如：

```
/* clear FMC bank0 operation error interrupt flag bit */

FlagStatus flag = fmc_interrupt_flag_clear (FMC_INT_FLAG_BANK0_PGERR);
```

函数 `fmc_bank0_state_get`

函数`fmc_bank0_state_get`描述见下表：

表 3-377. 函数 `fmc_bank0_state_get`

函数名称	<code>fmc_bank0_state_get</code>
函数原型	<code>fmc_state_enum fmc_bank0_state_get(void);</code>
功能描述	获取FMC BANK0状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
<code>fmc_state_enum</code>	FMC状态，详细请参考 枚举fmc_state_enum

例如：

```
/* get the FMC BANK0 state */

fmc_state_enum state = fmc_bank0_state_get();
```

函数 fmc_bank1_state_get

函数fmc_bank1_state_get描述见下表:

表 3-378. 函数 fmc_bank1_state_get

函数名称	fmc_bank1_state_get
函数原型	fmc_state_enum fmc_bank1_state_get(void);
功能描述	获取FMC BANK1状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* get the FMC BANK1 state */
fmc_state_enum state = fmc_bank1_state_get();
```

函数 fmc_bank0_ready_wait

函数 fmc_bank0_ready_wait描述见下表:

表 3-379. 函数 fmc_bank0_ready_wait

函数名称	fmc_bank0_ready_wait
函数原型	fmc_state_enum fmc_bank0_ready_wait(uint32_t timeout);
功能描述	检查FMC BANK0是否准备好
先决条件	-
被调用函数	fmc_bank0_state_get
输入参数{in}	
timeout	超时时长
输出参数{out}	

-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* check whether FMC BANK0 is ready or not */

fmc_state_enum state = fmc_bank0_ready_wait(0x00001000);
```

函数 fmc_bank1_ready_wait

函数 fmc_bank1_ready_wait描述见下表:

表 3-380. 函数 fmc_bank1_ready_wait

函数名称	fmc_bank1_ready_wait
函数原型	fmc_state_enum fmc_bank1_ready_wait(uint32_t timeout);
功能描述	检查FMC BANK1是否准备好
先决条件	-
被调用函数	fmc_bank1_state_get
输入参数{in}	
timeout	超时时长
输出参数{out}	
-	-
返回值	
fmc_state_enum	FMC状态, 详细请参考 枚举fmc_state_enum

例如:

```
/* check whether FMC BANK1 is ready or not */

fmc_state_enum state = fmc_bank1_ready_wait(0x00001000);
```

3.14. FWDGT

独立看门狗定时器（FWDGT）是一个硬件计时电路，用来监测由软件故障导致的系统故障。适合于需要独立环境且对计时精度要求不高的场合。章节[3.14.1](#)描述了FWDGT的寄存器列表，章节[3.14.2](#)对FWDGT库函数进行说明。

3.14.1. 外设寄存器说明

FWDGT寄存器列表如下表所示:

表 3-381. FWDGT 寄存器

寄存器名称	寄存器描述
FWDGT_CTL	控制寄存器
FWDGT_PSC	预分频寄存器
FWDGT_RLD	重装载寄存器
FWDGT_STAT	状态寄存器

3.14.2. 外设库函数说明

FWDGT库函数列表如下表所示:

表 3-382. FWDGT 库函数

库函数名称	库函数描述
fwdgt_write_enable	使能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_write_disable	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
fwdgt_enable	使能FWDGT
fwdgt_counter_reload	按照FWDGT_RLD寄存器的值重装载IWDG计数器
fwdgt_config	设置FWDGT重装载值、预分频值
fwdgt_flag_get	获取FWDGT标志位状态

函数 fwdgt_write_enable

函数fwdgt_write_enable描述见下表:

表 3-383. 函数 fwdgt_write_enable

函数名称	fwdgt_write_enable
函数原型	void fwdgt_write_enable(void);
功能描述	使能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable write access to FWDGT_PSC and FWDGT_RLD */

fwdgt_write_enable();
```

函数 **fwdgt_write_disable**

函数fwdgt_write_disable描述见下表：

表 3-384. 函数 **fwdgt_write_disable**

函数名称	fwdgt_write_disable
函数原型	void fwdgt_write_disable(void);
功能描述	失能对寄存器FWDGT_PSC和FWDGT_RLD的写操作
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable write access to FWDGT_PSC and FWDGT_RLD */

fwdgt_write_disable();
```

函数 **fwdgt_enable**

函数fwdgt_enable描述见下表：

表 3-385. 函数 fwdgt_enable

函数名称	fwdgt_enable
函数原型	void fwdgt_enable(void);
功能描述	使能FWDGT
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start the free watchdog timer counter */

fwdgt_enable();
```

函数 fwdgt_counter_reload

函数fwdgt_counter_reload描述见下表：

表 3-386. 函数 fwdgt_counter_reload

函数名称	fwdgt_counter_reload
函数原型	void fwdgt_counter_reload(void);
功能描述	按照FWDGT_RLD寄存器的值重装载IWDG计数器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* reload FWDGT counter */

fwdgt_counter_reload();
```

函数 **fwdgt_config**

函数fwdgt_config描述见下表：

表 3-387. 函数 **fwdgt_config**

函数名称	fwdgt_config
函数原型	ErrStatus fwdgt_config(uint16_t reload_value, uint8_t prescaler_div);
功能描述	设置FWDGT重装载值、预分频值
先决条件	-
被调用函数	-
输入参数{in}	
reload_value	重装载值(0x0000 - 0xFFFF)
输入参数{in}	
prescaler_div	FWDGT预分频值
FWDGT_PSC_DIV4	FWDGT预分频值设为4
FWDGT_PSC_DIV8	FWDGT预分频值设为8
FWDGT_PSC_DIV16	FWDGT预分频值设为16
FWDGT_PSC_DIV32	FWDGT预分频值设为32
FWDGT_PSC_DIV64	FWDGT预分频值设为64
FWDGT_PSC_DIV128	FWDGT prescaler set to 128
FWDGT_PSC_DIV256	FWDGT prescaler set to 256
输出参数{out}	
-	-

返回值	
ErrStatus	ERROR or SUCCESS

例如：

```
/* confiure FWDGT counter clock: 40KHz(IRC40K) / 64 = 0.625 KHz */
fwdgt_config(2*500, FWDGT_PSC_DIV64);
```

函数 fwdgt_flag_get

函数fwdgt_flag_get描述见下表：

表 3-388. 函数 fwdgt_flag_get

函数名称	fwdgt_flag_get
函数原型	FlagStatus fwdgt_flag_get(uint16_t flag);
功能描述	获取FWDGT标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	需要获取状态的FWDGT标志位
FWDGT_FLAG_PUD	预分频值更新进行中
FWDGT_FLAG_RUD	重装载值更新进行中
输出参数{out}	
-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if a prescaler value update is on going */
```

```
FlagStatus status;
status = fwdgt_flag_get (FWDGT_FLAG_PUD);
if(status == RESET)
{
    ...
}
```

```
    }else  
    {  
        ...  
    }
```

3.15. GPIO

GPIO用来实现各片上设备的逻辑输入/输出功能。章节[3.14.1](#)描述了GPIO的寄存器列表，章节[3.14.2](#)对GPIO库函数进行说明。

3.15.1. 外设寄存器说明

GPIO寄存器列表如下表所示：

表 3-389. GPIO 寄存器

寄存器名称	寄存器描述
GPIOx_CTL0	端口控制寄存器0
GPIOx_CTL1	端口控制寄存器1
GPIOx_ISTAT	端口输入状态寄存器
GPIOx_OCTL	端口输出控制寄存器
GPIOx_BOP	端口位操作寄存器
GPIOx_BC	位清除寄存器
GPIOx_LOCK	端口配置锁定寄存器
GPIOx_SPD	端口位速度寄存器
AFIO_EC	事件控制寄存器
AFIO_PCF0	AFIO端口配置寄存器0
AFIO_EXTI0	EXTI源选择寄存器0寄存器
AFIO_EXTI1	EXTI源选择寄存器1寄存器
AFIO_EXTI2	EXTI源选择寄存器2寄存器
AFIO_EXTI3	EXTI源选择寄存器3寄存器
AFIO_PCF1	AFIO端口配置寄存器1
AFIO_CPSCTL	IO补偿控制寄存器

3.15.2. 外设库函数说明

GPIO库函数列表如下表所示：

表 3-390. GPIO 库函数

库函数名称	库函数描述
gpio_deinit	复位外设GPIOx
gpio_afio_deinit	复位AFIO
gpio_init	GPIO参数初始化
gpio_bit_set	置位引脚值
gpio_bit_reset	复位引脚值
gpio_bit_write	将特定的值写入引脚
gpio_port_write	将特定的值写入一组端口
gpio_input_bit_get	获取引脚的输入值
gpio_input_port_get	获取一组端口的输入值
gpio_output_bit_get	获取引脚的输出值
gpio_output_port_get	获取一组端口的输出值
gpio_pin_remap_config	配置GPIO引脚重映射
gpio_ethernet_phy_select	以太网MII或RMII PHY选择（互联型）
gpio_exti_source_select	选择哪个引脚作为EXTI源
gpio_event_output_config	配置事件输出
gpio_event_output_enable	事件输出使能
gpio_event_output_disable	事件输出除能
gpio_pin_lock	相应的引脚配置被锁定
gpio_compensation_config	配置I/O补偿单元
gpio_compensation_flag_get	检测I/O补偿单元是否准备好

函数 gpio_deinit

函数gpio_deinit描述见下表：

表 3-391. 函数 gpio_deinit

函数名称	gpio_deinit

函数原型	void gpio_deinit(uint32_t gpio_periph);
功能描述	复位外设GPIOx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset GPIOA */
gpio_deinit(GPIOA);
```

函数 gpio_afio_deinit

函数gpio_afio_deinit描述见下表：

表 3-392. 函数 gpio_afio_deinit

函数名称	gpio_afio_deinit
函数原型	void gpio_afio_deinit(void);
功能描述	复位AFIO
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset alternate function */
```

```
gpio_afio_deinit();
```

函数 gpio_init

函数gpio_init描述见下表：

表 3-393. 函数 gpio_init

函数名称	gpio_init
函数原型	void gpio_init(uint32_t gpio_periph, uint32_t mode, uint32_t speed, uint32_t pin);
功能描述	GPIO参数初始化
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
gpio_mode	GPIO引脚模式
GPIO_MODE_AIN	模拟输入模式
GPIO_MODE_IN_FLOATING	浮空输入模式
GPIO_MODE_IPD	下拉输入模式
GPIO_MODE_IPU	上拉输入模式
GPIO_MODE_OUT_OD	开漏输出模式
GPIO_MODE_OUT_PP	推挽输出模式
GPIO_MODE_AF_OD	AFIO开漏输出模式
GPIO_MODE_AF_PP	AFIO推挽输出模式

输入参数{in}	
speed	GPIO输出最大速度
GPIO_OSPEED_10MHZ	最大速度为10MHZ
GPIO_OSPEED_2MHZ	最大速度为2MHZ
GPIO_OSPEED_50MHZ	最大速度为50MHZ
GPIO_OSPEED_MAX	最大速度大于50MHZ
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config PA0 as analog input mode*/
gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
```

函数 **gpio_bit_set**

函数gpio_bit_set描述见下表：

表 3-394. 函数 **gpio_bit_set**

函数名称	gpio_bit_set
函数原型	void gpio_bit_set(uint32_t gpio_periph, uint32_t pin);
功能描述	置位GPIO引脚值
先决条件	-
被调用函数	-

输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

函数 gpio_bit_reset

函数gpio_bit_reset描述见下表：

表 3-395. 函数 gpio_bit_reset

函数名称	gpio_bit_reset
函数原型	void gpio_bit_reset(uint32_t gpio_periph, uint32_t pin);
功能描述	复位GPIO引脚值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)

<code>GPIO_PIN_ALL</code>	所有引脚
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset PA0*/
gpio_bit_set(GPIOA, GPIO_PIN_0);
```

函数 `gpio_bit_write`

函数`gpio_bit_write`描述见下表：

表 3-396. 函数 `gpio_bit_write`

函数名称	<code>gpio_bit_write</code>
函数原型	<code>void gpio_bit_write(uint32_t gpio_periph, uint32_t pin, bit_status bit_value);</code>
功能描述	将特定的值写入GPIO引脚
先决条件	-
被调用函数	-
输入参数{in}	
<code>gpio_periph</code>	GPIO端口
<code>GPIOx</code>	端口选择 ($x=A,B,C,D,E,F,G$)
输入参数{in}	
<code>pin</code>	GPIO引脚
<code>GPIO_PIN_x</code>	引脚选择 ($x=0..15$)
<code>GPIO_PIN_ALL</code>	所有引脚
输入参数{in}	
<code>bit_value</code>	设置或清除
<code>RESET</code>	清除引脚值
<code>SET</code>	设置引脚值
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* write 1 to PA0*/
gpio_bit_write(GPIOA, GPIO_PIN_0, SET);
```

函数 **gpio_port_write**

函数gpio_port_write描述见下表：

表 3-397. 函数 **gpio_port_write**

函数名称	gpio_port_write
函数原型	void gpio_port_write(uint32_t gpio_periph, uint16_t data);
功能描述	将特定的值写入端口
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
data	将要写入的具体值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write 1010 0101 1010 0101 to Port A */
gpio_port_write(GPIOA, 0xA5A5);
```

函数 **gpio_input_bit_get**

函数gpio_input_bit_get描述见下表：

表 3-398. 函数 gpio_input_bit_get

函数名称	gpio_input_bit_get
函数原型	FlagStatus gpio_input_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取GPIO引脚的输入值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get status of PA0*/
FlagStatus bit_state;
bit_state = gpio_input_bit_get (GPIOA, GPIO_PIN_0);
```

函数 gpio_input_port_get

函数gpio_input_port_get描述见下表：

表 3-399. 函数 gpio_input_port_get

函数名称	gpio_input_port_get
函数原型	uint16_t gpio_input_port_get(uint32_t gpio_periph);
功能描述	获取GPIO端口的输入值
先决条件	-

被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如：

```
/* get input value of Port A */
uint16_t port_state;
port_state = gpio_input_bit_get(GPIOA);
```

函数 **gpio_output_bit_get**

函数gpio_output_bit_get描述见下表：

表 3-400. 函数 **gpio_output_bit_get**

函数名称	gpio_output_bit_get
函数原型	FlagStatus gpio_output_bit_get(uint32_t gpio_periph, uint32_t pin);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	

-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get output status of PA0 */
FlagStatus bit_state;
bit_state = gpio_output_bit_get (GPIOA, GPIO_PIN_0);
```

函数 gpio_output_port_get

函数gpio_output_port_get描述见下表：

表 3-401. 函数 gpio_output_port_get

函数名称	gpio_output_port_get
函数原型	uint16_t gpio_output_port_get(uint32_t gpio_periph);
功能描述	获取引脚的输出值
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输出参数{out}	
-	-
返回值	
uint16_t	0x0000-0xFFFF

例如：

```
/* get output value of Port A */
uint16_t port_state;
port_state = gpio_output_port_get (GPIOA);
```

函数 gpio_pin_remap_config

函数gpio_pin_remap_config描述见下表：

表 3-402. 函数 gpio_pin_remap_config

函数名称	gpio_pin_remap_config
函数原型	void gpio_pin_remap_config(uint32_t gpio_remap, ControlStatus newvalue);
功能描述	配置GPIO引脚重映射
先决条件	-
被调用函数	-
输入参数{in}	
gpio_remap	选择重映射
GPIO_SPI0_REMAP	SPI0重映射
GPIO_I2C0_REMAP	I2C0重映射
GPIO_USART0_REMAP	USART0重映射
GPIO_USART1_REMAP	USART1重映射
GPIO_USART2_PARTIAL_REMAP	USART2部分重映射
GPIO_USART2_FULL_REMAP	USART2全部重映射
GPIO_TIMER0_PARTIAL_REMAP	TIMER0部分重映射
GPIO_TIMER0_FULL_REMAP	TIMER0全部重映射
GPIO_TIMER1_PARTIAL_REMAP1	TIMER1部分重映射
GPIO_TIMER1_PARTIAL_REMAP2	TIMER1部分重映射
GPIO_TIMER1_FULL_REMAP	TIMER1全部重映射
GPIO_TIMER2_PARTIAL_REMAP	TIMER2部分重映射
GPIO_TIMER2_FULL_REMAP	TIMER2全部重映射

<code>GPIO_TIMER3_REMAP</code>	TIMER3重映射
<code>GPIO_CAN_PARTIAL_REMAP</code>	CAN部分重映射（高密度型和超高密度型）
<code>GPIO_CAN_FULL_REMAP</code>	CAN全部重映射（高密度型和超高密度型）
<code>GPIO_CAN0_PARTIAL_REMAP</code>	CAN0部分重映射（互联型）
<code>GPIO_CAN0_FULL_REMAP</code>	CAN0全部重映射（互联型）
<code>GPIO_PD01_REMAP</code>	PD01重映射
<code>GPIO_TIMER4CH3_IREMAP</code>	TIMER4通道3内部重映射（互联型和高密度型）
<code>GPIO_ADC0_ETRGENS_REMAP</code>	ADC0注入转换外部触发重映射（高密度型和超高密度型）
<code>GPIO_ADC0_ETRGREG_REMAP</code>	ADC0规则转换外部触发重映射（高密度型和超高密度型）
<code>GPIO_ADC1_ETRGENS_REMAP</code>	ADC1注入转换外部触发重映射（高密度型和超高密度型）
<code>GPIO_ADC1_ETRGREG_REMAP</code>	ADC1规则转换外部触发重映射（高密度型和超高密度型）
<code>GPIO_ENET_REMAP</code>	以太网重映射（互联型）
<code>GPIO_CAN1_REMAP</code>	CAN1重映射（互联型）
<code>GPIO_SWJ_NONJTRST_REMAP</code>	全部的SWJ(JTAG-DP + SW-DP)，但是不包括NJTRST
<code>GPIO_SWJ_SWDPE_NABLE_REMAP</code>	JTAG-DP除能，SW-DP使能
<code>GPIO_SWJ_DISABLE_REMAP</code>	JTAG-DP除能，SW-DP除能
<code>GPIO_SPI2_REMAP</code>	SPI2重映射（互联型）
<code>GPIO_TIMER1ITR0_REMAP</code>	TIMER1内部触发0重映射（互联型）
<code>GPIO_PTP_PPS</code>	以太网PTP PPS重映射（互联型）

<i>_REMAP</i>	
<i>GPIO_TIMER8_REMAP</i>	TIMER8重映射
<i>GPIO_TIMER9_REMAP</i>	TIMER9重映射
<i>GPIO_TIMER10_REMAP</i>	TIMER10重映射
<i>GPIO_TIMER12_REMAP</i>	TIMER12重映射
<i>GPIO_TIMER13_REMAP</i>	TIMER13重映射
<i>GPIO_EXMC_NADV_REMAP</i>	EXMC_NADV 连接/断开
<i>GPIO_CTC_REMAP0</i>	CTC重映射(PD15)
<i>GPIO_CTC_REMAP1</i>	CTC重映射(PF0)
输入参数{in}	
<i>newvalue</i>	是否使能
<i>ENABLE</i>	使能
<i>DISABLE</i>	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 remapping */
gpio_pin_remap_config (GPIO_SPI0_REMAP, ENABLE);
```

函数 **gpio_ethernet_phy_select**

函数gpio_ethernet_phy_select描述见下表：

表 3-403. 函数 **gpio_ethernet_phy_select**

函数名称	gpio_ethernet_phy_select
------	--------------------------

函数原型	void gpio_ethernet_phy_select(uint32_t enet_sel);
功能描述	以太网MII或RMII PHY选择（互联型）
先决条件	-
被调用函数	-
输入参数{in}	
enet_sel	以太网MII和RMII PHY选择
GPIO_ENET_PHY_MII	选择MII
GPIO_ENET_PHY_RMII	选择RMII
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure ethernet MAC for connection with an RMII PHY */
gpio_ethernet_phy_select(GPIO_ENET_PHY_RMII);
```

函数 gpio_exti_source_select

函数gpio_exti_source_select描述见下表：

表 3-404. 函数 gpio_exti_source_select

函数名称	gpio_exti_source_select
函数原型	void gpio_exti_source_select(uint8_t output_port, uint8_t output_pin);
功能描述	选择哪个引脚作为EXTI源
先决条件	-
被调用函数	-
输入参数{in}	
output_port	EXTI源端口
GPIO_PORT_SOURCE_GPIOn	EXTI源端口选择 (x=A,B,C,D,E,F,G)

输入参数{in}	
output_pin	源端口引脚
GPIO_PIN_SOURCE_x	源端口引脚选择 (x=0..15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config PA0 as EXTI source */
gpio_exti_source_select(GPIO_PORT_SOURCE_GPIOA, GPIO_PIN_SOURCE_0);
```

函数 **gpio_event_output_config**

函数gpio_event_output_config描述见下表：

表 3-405. 函数 **gpio_event_output_config**

函数名称	gpio_event_output_config
函数原型	void gpio_event_output_config(uint8_t output_port, uint8_t output_pin);
功能描述	配置事件输出
先决条件	-
被调用函数	-
输入参数{in}	
output_port	GPIO事件输出端口
GPIO_EVENT_PORT_GPIOx	事件输出端口选择 (x = A,B,C,D,E)
输入参数{in}	
output_pin	GPIO事件输出引脚
GPIO_EVENT_PIN_x	事件引脚选择 (x=0..15)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure PA0 as the output of event */

gpio_event_output_config(GPIO_EVENT_PORT_GPIOA, GPIO_EVENT_PIN_0);
```

函数 **gpio_event_output_enable**

函数gpio_event_output_enable描述见下表：

表 3-406. 函数 **gpio_event_output_enable**

函数名称	gpio_event_output_enable
函数原型	void gpio_event_output_enable(void);
功能描述	事件输出使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable GPIO pin event output */

gpio_event_output_enable();
```

函数 **gpio_event_output_disable**

函数gpio_event_output_disable描述见下表：

表 3-407. 函数 **gpio_event_output_disable**

函数名称	gpio_event_output_disable
函数原型	void gpio_event_output_disable(void);
功能描述	事件输出禁能

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable GPIO pin event output */
gpio_event_output_disable();
```

函数 gpio_pin_lock

函数gpio_pin_lock描述见下表：

表 3-408. 函数 gpio_pin_lock

函数名称	gpio_pin_lock
函数原型	void gpio_pin_lock(uint32_t gpio_periph,uint32_t pin);
功能描述	相应的引脚配置被锁定
先决条件	-
被调用函数	-
输入参数{in}	
gpio_periph	GPIO端口
GPIOx	端口选择 (x=A,B,C,D,E,F,G)
输入参数{in}	
pin	GPIO引脚
GPIO_PIN_x	引脚选择 (x=0..15)
GPIO_PIN_ALL	所有引脚
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* lock PA0 */
gpio_pin_lock(GPIOA, GPIO_PIN_0);
```

函数 **gpio_compensation_config**

函数gpio_compensation_config描述见下表：

表 3-409. 函数 **gpio_compensation_config**

函数名称	
	gpio_compensation_config
函数原型	void gpio_compensation_config(uint32_t compensation);
功能描述	配置I/O补偿单元
先决条件	-
被调用函数	-
输入参数{in}	
compensation	指定I/O补偿单元模式
GPIO_COMPENSATION_ENABLE	I/O补偿单元使能
GPIO_COMPENSATION_DISABLE	I/O补偿单元禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enabled I/O compensation cell */
gpio_compensation_config(GPIO_COMPENSATION_ENABLE);
```

函数 **gpio_compensation_flag_get**

函数gpio_compensation_flag_get描述见下表：

表 3-410. 函数 gpio_compensation_flag_get

函数名称	gpio_compensation_flag_get
函数原型	FlagStatus gpio_compensation_flag_get(void);
功能描述	检测I/O补偿单元是否准备好
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* check the I/O compensation cell state */

FlagStatus cell_state;

cell_state = gpio_compensation_flag_get (void);
```

3.16. I2C

I2C（内部集成电路总线）模块提供了符合工业标准的两线串行制接口，可用于MCU和外部I2C设备的通讯。章节[3.15.1](#)描述了I2C的寄存器列表，章节[3.15.2](#)对I2C库函数进行说明。

3.16.1. 外设寄存器说明

I2C寄存器列表如下表所示：

表 3-411. I2C 寄存器

寄存器名称	寄存器描述
I2C_CTL0	控制寄存器0
I2C_CTL1	控制寄存器1
I2C_SADDR0	从机地址寄存器0
I2C_SADDR1	从机地址寄存器1

寄存器名称	寄存器描述
I2C_DATA	传输缓冲区寄存器
I2C_STAT0	传输状态寄存器0
I2C_STAT1	传输状态寄存器1
I2C_CKCFG	时钟配置寄存器
I2C_RT	上升时间寄存器
I2C_FMPCFG	快速+ 模式配置寄存器

3.16.2. 外设库函数说明

I2C库函数列表如下表所示:

表 3-412. I2C 库函数

库函数名称	库函数描述
i2c_deinit	复位外设I2C
i2c_clock_config	配置I2C时钟
i2c_mode_addr_config	配置I2C地址
i2c_smbus_type_config	SMBus类型选择
i2c_ack_config	是否发送ACK
i2c_ackpos_config	ACK位置配置
i2c_master_addressing	主机发送从机地址
i2c_dualaddr_enable	双地址模式使能
i2c_enable	使能I2C模块
i2c_disable	关闭I2C模块
i2c_start_on_bus	在I2C总线上生成起始位
i2c_stop_on_bus	在I2C总线上生成停止位
i2c_data_transmit	发送数据
i2c_data_receive	接收数据
i2c_dma_enable	I2C DMA模式使能
i2c_dma_last_transfer_config	配置下一个DMA EOT是最后传输
i2c_stretch_scl_low_config	当从机数据没有准备好时是否拉低SCL

库函数名称	库函数描述
i2c_slave_response_to_gcall_config	从机是否响应广播呼叫
i2c_software_reset_config	配置I2C软件复位
i2c_pec_enable	报文错误校验使能
i2c_pec_transfer_enable	传输PEC值使能
i2c_pec_value_get	获取报文错误校验值
i2c_smbus_issue_alert	通过SMBA引脚发送警告
i2c_smbus_arp_enable	SMBus下ARP协议是否开启
i2c_flag_get	标志位获取
i2c_flag_clear	清除标志位
i2c_interrupt_enable	中断使能
i2c_interrupt_disable	中断除能
i2c_interrupt_flag_get	中断标志位获取
i2c_interrupt_flag_clear	中断标志位清除

函数 i2c_deinit

函数i2c_deinit描述见下表：

表 3-413. 函数 i2c_deinit

函数名称	i2c_deinit
函数原型	void i2c_deinit(uint32_t i2c_periph);
功能描述	复位外设I2C
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* reset I2C0 */
```

```
i2c_deinit(I2C0);
```

函数 i2c_clock_config

函数i2c_clock_config描述见下表：

表 3-414. 函数 i2c_clock_config

函数名称	i2c_clock_config
函数原型	void i2c_clock_config(uint32_t i2c_periph, uint32_t clkspeed, uint32_t dutycyc);
功能描述	配置I2C时钟
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
clkspeed	i2c时钟速率
输入参数{in}	
dutycyc	快速模式下占空比
I2C_DTCY_2	T_low/T_high=2
I2C_DTCY_16_9	T_low/T_high=16/9
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 clock speed as 100KHz*/
```

```
i2c_clock_config(I2C0, 100000, I2C_DTCY_2);
```

函数 i2c_mode_addr_config

函数i2c_mode_addr_config描述见下表:

表 3-415. 函数 i2c_mode_addr_config

函数名称	i2c_mode_addr_config
函数原型	void i2c_mode_addr_config(uint32_t i2c_periph, uint32_t mode, uint32_t addformat, uint32_t addr);
功能描述	配置I2C地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
i2cmod	模式选择
I2C_I2CMODE_ENABLE	I2C 模式
I2C_SMBUSMODE_ENABLE	SMBus 模式
输入参数{in}	
addformat	7bits 或 10bits
I2C_ADDFORMAT_7BITS	7bits
I2C_ADDFORMAT_10BITS	10bits
输入参数{in}	
addr	I2C地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure I2C0 address as 0x82, using 7 bits */

i2c_mode_addr_config(I2C0, I2C_I2CMODE_ENABLE, I2C_ADDFORMAT_7BITS, 0x82);
```

函数 i2c_smbus_type_config

函数i2c_smbus_type_config描述见下表：

表 3-416. 函数 i2c_smbus_type_config

函数名称	i2c_smbus_type_config
函数原型	void i2c_smbus_type_config(uint32_t i2c_periph, uint32_t type);
功能描述	SMBus类型选择
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
type	主机或从机
I2C_SMBUS_DEVIC CE	从机
I2C_SMBUS_HOST	主机
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* config I2C0 as SMBUS host type */

i2c_smbus_type_config (I2C0, I2C_SMBUS_HOST);
```

函数 i2c_ack_config

函数i2c_ack_config描述见下表：

表 3-417. 函数 i2c_ack_config

函数名称	i2c_ack_config
函数原型	void i2c_ack_config(uint32_t i2c_periph, uint32_t ack);
功能描述	是否发送ACK
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
ack	是否发送ACK
I2C_ACK_ENABLE	ACK 会被发送
I2C_ACK_DISABLE	ACK 不会发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 will sent ACK */
i2c_ack_config (I2C0, I2C_ACK_ENABLE);
```

函数 i2c_ackpos_config

函数i2c_ackpos_config描述见下表：

表 3-418. 函数 i2c_ackpos_config

函数名称	i2c_ackpos_config
函数原型	void i2c_ackpos_config(uint32_t i2c_periph, uint32_t pos);
功能描述	ACK位置配置
先决条件	-
被调用函数	-

输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
pos	ACK位置
I2C_ACKPOS_CURRENT	当前正在接收的字节是否发送ACK
I2C_ACKPOS_NEXT	下一个接收的字节是否发送ACK
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*设The ACK of I2C0 is send for the current frame */
i2c_ackpos_config(I2C0, I2C_ACKPOS_CURRENT);
```

函数 i2c_master_addressing

函数i2c_master_addressing描述见下表:

表 3-419. 函数 i2c_master_addressing

函数名称	i2c_master_addressing
函数原型	void i2c_master_addressing(uint32_t i2c_periph, uint32_t addr, uint32_t trandirection);
功能描述	主机发送从机地址
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	

addr	从机地址
输入参数{in}	
trandirection	发送或接收
<i>I2C_TRANSMITTER</i>	发送
<i>I2C_RECEIVER</i>	接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* send slave address to I2C bus and I2C0 act as receiver */
i2c_master_addressing(I2C0, I2C1_SLAVE_ADDRESS7, I2C_RECEIVER);
```

函数 i2c_dualaddr_enable

函数i2c_dualaddr_enable描述见下表：

表 3-420. 函数 i2c_dualaddr_enable

函数名称	i2c_dualaddr_enable
函数原型	void i2c_dualaddr_enable(uint32_t i2c_periph, uint32_t dualaddr);
功能描述	双地址模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
dualaddr	是否使能
<i>I2C_DUADEN_DISABLE</i>	不使能双地址模式
<i>I2C_DUADEN_ENABLE</i>	使能双地址模式

BLE	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 dual-address */
i2c_dualaddr_enable (I2C0, I2C_DUADEN_ENABLE);
```

函数 i2c_enable

函数i2c_enable描述见下表：

表 3-421. 函数 i2c_enable

函数名称	i2c_enable
函数原型	void i2c_enable(uint32_t i2c_periph);
功能描述	使能I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 */
i2c_enable (I2C0);
```

函数 i2c_disable

函数i2c_disable描述见下表：

表 3-422. 函数 i2c_disable

函数名称	i2c_disable
函数原型	void i2c_disable(uint32_t i2c_periph);
功能描述	关闭I2C模块
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 */
i2c_disable (I2C0);
```

函数 i2c_start_on_bus

函数i2c_start_on_bus描述见下表：

表 3-423. 函数 i2c_start_on_bus

函数名称	i2c_start_on_bus
函数原型	void i2c_start_on_bus(uint32_t i2c_periph);
功能描述	在I2C总线上生成起始位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* I2C0 send a start condition to I2C bus */
i2c_start_on_bus (I2C0);
```

函数 i2c_stop_on_bus

函数i2c_stop_on_bus描述见下表：

表 3-424. 函数 i2c_stop_on_bus

函数名称	i2c_stop_on_bus
函数原型	void i2c_stop_on_bus(uint32_t i2c_periph);
功能描述	在I2C总线上生成停止位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 generate a STOP condition to I2C bus */
i2c_stop_on_bus (I2C0);
```

函数 i2c_data_transmit

函数i2c_data_transmit描述见下表：

表 3-425. 函数 i2c_data_transmit

函数名称	i2c_data_transmit
------	-------------------

函数原型	void i2c_data_transmit(uint32_t i2c_periph, uint8_t data);
功能描述	发送数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
data	传输的数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transmit data */
```

```
i2c_data_transmit (I2C0);
```

函数 i2c_data_receive

函数i2c_data_receive描述见下表：

表 3-426. 函数 i2c_data_receive

函数名称	i2c_data_receive
函数原型	uint8_t i2c_data_receive(uint32_t i2c_periph);
功能描述	接收数据
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	

-	-
返回值	
uint8_t	0x00..0xFF

例如：

```
/* I2C0 receive data */

uint8_t i2c_receiver;

i2c_receiver = i2c_data_receive (I2C0);
```

函数 i2c_dma_enable

函数i2c_dma_enable描述见下表：

表 3-427. 函数 i2c_dma_enable

函数名称	i2c_dma_enable
函数原型	void i2c_dma_enable(uint32_t i2c_periph, uint32_t dmastate);
功能描述	I2C DMA模式使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmastate	开启或关闭
I2C_DMA_ON	DMA模式开启
I2C_DMA_OFF	DMA模式关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 DMA mode enable */
```

```
i2c_dma_enable(I2C0, I2C_DMA_ON);
```

函数 i2c_dma_last_transfer_enable

函数i2c_dma_last_transfer_enable描述见下表:

表 3-428. 函数 i2c_dma_last_transfer_enable

函数名称	i2c_dma_last_transfer_enable
函数原型	void i2c_dma_last_transfer_enable(uint32_t i2c_periph, uint32_t dmalast);
功能描述	使能下一个DMA EOT是最后传输
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
dmalast	是否使能下一个DMA EOT是最后传输
I2C_DMALST_ON	使能
I2C_DMALST_OFF	不使能
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* next DMA EOT is the last transfer */

i2c_dma_last_transfer_enable(I2C0, I2C_DMALST_ON);
```

函数 i2c_stretch_scl_low_config

函数i2c_stretch_scl_low_config描述见下表:

表 3-429. 函数 i2c_stretch_scl_low_config

函数名称	i2c_stretch_scl_low_config
函数原型	void i2c_stretch_scl_low_config(uint32_t i2c_periph, uint32_t stretchpara);

功能描述	当从机数据没有准备好时是否拉低SCL
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
stretchpara	是否拉低SCL
I2C_SCLSTRETCH_ENABLE	不拉低SCL
I2C_SCLSTRETCH_DISABLE	不拉低SCL
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* stretch SCL low when data is not ready in slave mode */

i2c_stretch_scl_low_config (I2C0, I2C_SCLSTRETCH_ENABLE);
```

函数 i2c_slave_response_to_gcall_config

函数i2c_slave_response_to_gcall_config描述见下表：

表 3-430. 函数 i2c_slave_response_to_gcall_config

函数名称	i2c_slave_response_to_gcall_config
函数原型	void i2c_slave_response_to_gcall_config(uint32_t i2c_periph, uint32_t gcallpara);
功能描述	从机是否响应广播呼叫
先决条件	-
被调用函数	-
输入参数{in}	

i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
gcallpara	是否响应广播呼叫
<i>I2C_GCEN_ENABLE</i>	从机响应广播呼叫
<i>I2C_GCEN_DISABLE</i>	从机不响应广播呼叫
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 will response to a general call */
i2c_slave_response_to_gcall_config (I2C0, I2C_GCEN_ENABLE);
```

函数 i2c_software_reset_config

函数i2c_software_reset_config描述见下表：

表 3-431. 函数 i2c_software_reset_config

函数名称	i2c_software_reset_config
函数原型	void i2c_software_reset_config(uint32_t i2c_periph, uint32_t sreset);
功能描述	配置I2C软件复位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
sreset	是否复位
<i>I2C_SRESET_SET</i>	复位

<i>I2C_SRESET_RES</i> <i>ET</i>	没有复位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* software reset I2C0*/
i2c_software_reset_config(I2C0, I2C_SRESET_SET);
```

函数 i2c_pec_enable

函数i2c_pec_enable描述见下表：

表 3-432. 函数 i2c_pec_enable

函数名称	i2c_pec_enable
函数原型	void i2c_pec_enable(uint32_t i2c_periph, uint32_t pecstate);
功能描述	报文错误校验使能
先决条件	-
被调用函数	-
输入参数{in}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<i>pecpara</i>	开启或关闭
<i>I2C_PEC_ENABLE</i>	报文错误校验使能
<i>I2C_PEC_DISABLE</i>	报文错误校验关闭
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C PEC calculation */

i2c_pec_enable (I2C0, I2C_PEC_ENABLE);
```

函数 i2c_pec_transfer_enable

函数i2c_pec_transfer_enable描述见下表：

表 3-433. 函数 i2c_pec_transfer_enable

函数名称	i2c_pec_transfer_enable
函数原型	void i2c_pec_transfer_enable(uint32_t i2c_periph, uint32_t pecpara);
功能描述	传输PEC值使能
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
pecpara	是否传输PEC
I2C_PECTRANS_ENABLE	传输PEC
I2C_PECTRANS_DISABLE	不传输PEC
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 transfer PEC */

i2c_pec_transfer_enable (I2C0, I2C_PECTRANS_ENABLE);
```

函数 i2c_pec_value_get

函数i2c_pec_value_get描述见下表：

表 3-434. 函数 i2c_pec_value_get

函数名称	i2c_pec_value_get
函数原型	uint8_t i2c_pec_value_get(uint32_t i2c_periph);
功能描述	获取报文错误校验值
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输出参数{out}	
-	-
返回值	
uint8_t	PEC值

例如：

```
/* I2C0 get packet error checking value */
uint8_t pec_value;
pec_value = i2c_pec_value_get (I2C0);
```

函数 i2c_smbus_issue_alert

函数i2c_smbus_issue_alert描述见下表：

表 3-435. 函数 i2c_smbus_issue_alert

函数名称	i2c_smbus_issue_alert
函数原型	void i2c_smbus_issue_alert(uint32_t i2c_periph, uint32_t smbuspara);
功能描述	通过SMBA引脚发送警告
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)

输入参数{in}	
smbuspara	是否通过SMBA引脚发送警告
I2C_SALTSEND_ENABLE	通过SMBA引脚发送警告
I2C_SALTSEND_DISABLE	不通过SMBA引脚发送警告
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* I2C0 issue alert through SMBA pin enable */
i2c_smbus_issue_alert(I2C0, I2C_SALTSEND_ENABLE);
```

函数 i2c_smbus_arp_enable

函数i2c_smbus_arp_enable描述见下表：

表 3-436. 函数 i2c_smbus_arp_enable

函数名称	i2c_smbus_arp_enable
函数原型	void i2c_smbus_arp_enable(uint32_t i2c_periph, uint32_t arpstate);
功能描述	SMBus下ARP协议是否开启
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
arpstate	SMBus下ARP协议是否开启
I2C_ARP_ENABLE	使能ARP
I2C_ARP_DISABLE	关闭ARP
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable I2C0 ARP protocol in SMBus switch */

i2c_smbus_arp_enable (I2C0, I2C_ARP_ENABLE);
```

函数 i2c_flag_get

函数i2c_flag_get描述见下表：

表 3-437. 函数 i2c_flag_get

函数名称	i2c_flag_get
函数原型	FlagStatus i2c_flag_get(uint32_t i2c_periph,uint32_t flag);
功能描述	标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	需要获取的标志位
I2C_FLAG_SBSEN_D	起始位是否发送
I2C_FLAG_ADDSEND	主机模式下地址是否发送/从机模式下地址是否匹配
I2C_FLAG_BTC	字节传输完成
I2C_FLAG_ADD10SEND	主机模式下10位地址地址头发送完成
I2C_FLAG_STPDET	从机模式下监测到STOP结束位
I2C_FLAG_RBNE	接收期间I2C_DATA非空

<i>I2C_FLAG_TBE</i>	发送期间I2C_DATA为空
<i>I2C_FLAG_BERR</i>	总线错误，表示I2C总线上发生了预料之外的START起始位或STOP结束位
<i>I2C_FLAG_LOSTA_RB</i>	主机模式下仲裁丢失
<i>I2C_FLAG_AERR</i>	应答错误
<i>I2C_FLAG_OUERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_FLAG_PECER_R</i>	接收数据时PEC错误
<i>I2C_FLAG_SMBTO</i>	SMBus模式下超时信号
<i>I2C_FLAG_SMBAL_T</i>	SMBus警报状态
<i>I2C_FLAG_MASTERR</i>	表明I2C时钟在主机模式还是从机模式的标志位
<i>I2C_FLAG_I2CBSY</i>	忙标志
<i>I2C_FLAG_TRS</i>	I2C作发送端还是接收端
<i>I2C_FLAG_RXGC</i>	是否接收到广播地址(00h)
<i>I2C_FLAG_DEFSM_B</i>	从机模式下SMBus主机地址头
<i>I2C_FLAG_HSTSM_B</i>	从机模式下监测到SMBus主机地址头
<i>I2C_FLAG_DUMOD</i>	从机模式下双标志位表明哪个地址和双地址模式匹配
<i>I2C_FLAG_TFF</i>	发送帧下降沿标志
<i>I2C_FLAG_TFR</i>	发送帧上升沿标志
<i>I2C_FLAG_RFF</i>	接收帧下降沿标志
<i>I2C_FLAG_RFR</i>	接收帧上升沿标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```

/* check whether start condition send out */

FlagStatus flag_state = RESET;

flag_state = i2c_flag_get (I2C0, I2C_FLAG_SBSEND);

```

函数 i2c_flag_clear

函数i2c_flag_clear描述见下表:

表 3-438. 函数 i2c_flag_clear

函数名称	i2c_flag_clear
函数原型	void i2c_flag_clear(uint32_t i2c_periph, uint32_t flag);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
flag	标志位类型
I2C_FLAG_SMBAL_T	SMBus警报状态
I2C_FLAG_SMBTO	SMBus模式下超时信号
I2C_FLAG_PECER_R	接收数据时PEC错误
I2C_FLAG_OUERR	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
I2C_FLAG_AERR	应答错误
I2C_FLAG_LOSTA_RB	主机模式下仲裁丢失
I2C_FLAG_BERR	总线错误
I2C_FLAG_ADDSEN_D	主机模式下地址是否发送/从机模式下地址是否匹配，通过读I2C_STAT0和I2C_STAT1来清除
I2C_FLAG_TFF	发送帧下降沿标志
I2C_FLAG_TFR	发送帧上升沿标志

<i>I2C_FLAG_RFF</i>	接收帧下降沿标志
<i>I2C_FLAG_RFR</i>	接收帧上升沿标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear a bus error flag*/
i2c_flag_clear(I2C0, I2C_FLAG_BERR);
```

函数 i2c_interrupt_enable

函数i2c_interrupt_enable描述见下表：

表 3-439. 函数 i2c_interrupt_enable

函数名称	i2c_interrupt_enable
函数原型	void i2c_interrupt_enable(uint32_t i2c_periph, uint32_t inttype);
功能描述	中断使能
先决条件	-
被调用函数	-
输入参数{in}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<i>inttype</i>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能
<i>I2C_INT_TFF</i>	发送帧下降沿中断使能
<i>I2C_INT_TFR</i>	发送帧上升沿中断使能
<i>I2C_INT_RFF</i>	接收帧下降沿中断使能

<i>I2C_INT_RFR</i>	接收帧上升沿中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable I2C0 error interrupt */

i2c_interrupt_enable (I2C0, I2C_INT_EV);
```

函数 i2c_interrupt_disable

函数*i2c_interrupt_disable*描述见下表：

表 3-440. 函数 *i2c_interrupt_disable*

函数名称	<i>i2c_interrupt_disable</i>
函数原型	void i2c_interrupt_disable(uint32_t i2c_periph, uint32_t inttype);
功能描述	中断除能
先决条件	-
被调用函数	-
输入参数{in}	
<i>i2c_periph</i>	I2C外设
<i>I2Cx</i>	(x=0,1)
输入参数{in}	
<i>inttype</i>	中断类型
<i>I2C_INT_ERR</i>	错误中断使能
<i>I2C_INT_EV</i>	事件中断使能
<i>I2C_INT_BUF</i>	缓冲区中断使能
<i>I2C_INT_TFF</i>	发送帧下降沿中断使能
<i>I2C_INT_TFR</i>	发送帧上升沿中断使能
<i>I2C_INT_RFF</i>	接收帧下降沿中断使能
<i>I2C_INT_RFR</i>	接收帧上升沿中断使能

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2C0 error interrupt */

i2c_interrupt_disable (I2C0, I2C_INT_EV);
```

函数 i2c_interrupt_flag_get

函数i2c_interrupt_flag_get描述见下表：

表 3-441. 函数 i2c_interrupt_flag_get

函数名称	i2c_interrupt_flag_get
函数原型	FlagStatus i2c_interrupt_flag_get(uint32_t i2c_periph, uint32_t intflag);
功能描述	中断标志位获取
先决条件	-
被调用函数	-
输入参数{in}	
i2c_periph	I2C外设
I2Cx	(x=0,1)
输入参数{in}	
int_flag	中断标志
I2C_INT_FLAG_SB_SEND	主机模式下发送START起始位
I2C_INT_FLAG_ADDRESSSEND	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
I2C_INT_FLAG_BYTEC	字节发送结束
I2C_INT_FLAG_ADDRESS10SEND	主机模式下10位地址地址头被发送
I2C_INT_FLAG_STOPDET	从机模式下监测到STOP结束位

<i>I2C_INT_FLAG_RB NE</i>	接收期间I2C_DATA非空
<i>I2C_INT_FLAG_TB E</i>	发送期间I2C_DATA为空
<i>I2C_INT_FLAG_BE RR</i>	总线错误
<i>I2C_INT_FLAG_LO STARB</i>	主机模式下仲裁丢失
<i>I2C_INT_FLAG_AE RR</i>	应答错误
<i>I2C_INT_FLAG_OU ERR</i>	当禁用SCL拉低功能后，在从机模式下发生了过载或欠载事件
<i>I2C_INT_FLAG_PE CERR</i>	接收数据时PEC错误
<i>I2C_INT_FLAG_SM BTO</i>	SMBus模式下超时信号
<i>I2C_INT_FLAG_SM BALT</i>	SMBus警报状态
<i>I2C_INT_FLAG_TF F</i>	发送帧下降沿中断标志位
<i>I2C_INT_FLAG_TF R</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RF F</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RF R</i>	接收帧上升沿中断标志位
输出参数{out}	
-	-
返回值	
FlagStatus	SET / RESET

例如：

```
/* check the byte transmission finishes interrupt flag is set or not */
```

```
FlagStatus flag_state = RESET;
```

```
flag_state = i2c_interrupt_flag_get (I2C0, I2C_INT_FLAG_BTC);
```

函数 `i2c_interrupt_flag_clear`

函数*i2c_interrupt_flag_clear*描述见下表：

表 3-442. 函数 `i2c_interrupt_flag_clear`

函数名称	i2c_interrupt_flag_clear
函数原型	void i2c_interrupt_flag_clear(uint32_t i2c_periph, uint32_t intflag);
功能描述	中断标志位清除
先决条件	-
被调用函数	-
输入参数{in}	
<code>i2c_periph</code>	I2C外设
<code>I2Cx</code>	(x=0,1)
输入参数{in}	
<code>int_flag</code>	中断标志
<code>I2C_INT_FLAG_ADSEND</code>	主机模式下成功发送了地址 / 从机模式下接收到了地址并且和自身的地址匹配
<code>I2C_INT_FLAG_BER</code>	总线错误
<code>I2C_INT_FLAG_LOSTARB</code>	主机模式下仲裁丢失
<code>I2C_INT_FLAG_AER</code>	应答错误
<code>I2C_INT_FLAG_OVERRUN</code>	当禁用SCL 拉低功能后，在从机模式下发生了过载或欠载事件
<code>I2C_INT_FLAG_PECERR</code>	接收数据时PEC错误
<code>I2C_INT_FLAG_SMBUS_TIMEOUT</code>	SMBus模式下超时信号
<code>I2C_INT_FLAG_SMBUS_ALERT</code>	SMBus警报状态
<code>I2C_INT_FLAG_TRANSITION</code>	发送帧下降沿中断标志位

<i>F</i>	
<i>I2C_INT_FLAG_TF</i> <i>R</i>	发送帧上升沿中断标志位
<i>I2C_INT_FLAG_RF</i> <i>F</i>	接收帧下降沿中断标志位
<i>I2C_INT_FLAG_RF</i> <i>R</i>	接收帧上升沿中断标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the acknowledge error interrupt flag */
i2c_interrupt_flag_clear(I2C0, I2C_INT_FLAG_AERR);
```

3.17. MISC

MISC 是对嵌套向量中断控制器 (NVIC) 和系统定时器 (SysTick) 操作的软件包。章节 [3.17.1](#) 描述了 NVIC 和 SysTick 的寄存器列表，章节 [3.17.2](#) 对 MISC 库函数进行说明。

3.17.1. 外设寄存器说明

表 3-443. NVIC 寄存器

寄存器名称	寄存器描述
ISER ⁽¹⁾	中断使能寄存器
ICER ⁽¹⁾	中断禁能寄存器
ISPR ⁽¹⁾	中断挂起寄存器
ICPR ⁽¹⁾	中断清除寄存器
IABR ⁽¹⁾	中断活动状态寄存器
IP ⁽¹⁾	中断优先级寄存器
STIR ⁽¹⁾	软触发中断寄存器

寄存器名称	寄存器描述
CPUID ⁽²⁾	CPUID寄存器
ICSR ⁽²⁾	中断控制及状态寄存器
VTOR ⁽²⁾	向量表偏移量寄存器
AIRCR ⁽²⁾	应用程序中断及复位控制寄存器
SCR ⁽²⁾	系统控制寄存器
CCR ⁽²⁾	配置与控制寄存器
SHP ⁽²⁾	系统异常优先级寄存器
SHCSR ⁽²⁾	系统异常控制及状态寄存器
CFSR ⁽²⁾	配置错误状态寄存器
HFSR ⁽²⁾	硬错误状态寄存器
DFSR ⁽²⁾	调试错误状态寄存器
MMFAR ⁽²⁾	存储管理错误地址寄存器
BFAR ⁽²⁾	总线错误地址寄存器
AFSR ⁽²⁾	辅助错误地址寄存器
PFR ⁽²⁾	处理器特性寄存器
DFR ⁽²⁾	调试特性寄存器
ADR ⁽²⁾	辅助特性寄存器
MMFR ⁽²⁾	存储模型特性寄存器
ISAR ⁽²⁾	指令设置属性寄存器
CPACR ⁽²⁾	协处理器访问控制寄存器

1. 参考 core_cm4.h 文件中定义的结构体类型 NVIC_Type

2. 参考 core_cm4.h 文件中定义的结构体类型 SCB_Type

表 3-444. Systick 寄存器

寄存器名称	寄存器描述
CTRL ⁽¹⁾	系统控制和状态寄存器
LOAD ⁽¹⁾	系统重载值寄存器
VAL ⁽¹⁾	系统当前值寄存器
CALIB ⁽¹⁾	系统校准寄存器

1. 参考 core_cm4.h 文件中定义的结构体类型 SysTick_Type

3.17.2. 外设库函数说明

枚举类型 IRQn_Type

表 3-445. 枚举类型 IRQn_Type

成员名称	功能描述
WWDGT_IRQHandler	窗口看门狗中断
LVD_IRQHandler	连接到 EXTI 线的 LVD 中断
TAMPER_IRQHandler	侵入检测中断
RTC_IRQHandler	RTC 全局中断
FMC_IRQHandler	FMC 全局中断
RCU_CTC_IRQHandler	RCU 和 CTC 全局中断
EXTI0_IRQHandler	EXTI 线 0 中断
EXTI1_IRQHandler	EXTI 线 1 中断
EXTI2_IRQHandler	EXTI 线 2 中断
EXTI3_IRQHandler	EXTI 线 3 中断
EXTI4_IRQHandler	EXTI 线 4 中断
DMA0_Channel0_IRQHandler	DMA0 通道 0 全局中断
DMA0_Channel1_IRQHandler	DMA0 通道 1 全局中断
DMA0_Channel2_IRQHandler	DMA0 通道 2 全局中断
DMA0_Channel3_IRQHandler	DMA0 通道 3 全局中断
DMA0_Channel4_IRQHandler	DMA0 通道 4 全局中断
DMA0_Channel5_IRQHandler	DMA0 通道 5 全局中断
DMA0_Channel6_IRQHandler	DMA0 通道 6 全局中断
ADC0_1_IRQHandler	ADC0 和 ADC1 全局中断
USBD_HP_CAN0_TX_IRQHandler / CAN0_TX_IRQHandler	USBD 高优先级 CAN0 发送中断 / CAN0 发送中断
USBD_LP_CAN0_RX0_IRQHandler / CAN0_RX0_IRQHandler	USBD 低优先级 CAN0 接收 0 中断 / CAN0 接收 0 中断
CAN0_RX1_IRQHandler	CAN0 接收 1 中断
CAN0_EWMC_IRQHandler	CAN0 EWMC 中断
EXTI5_9_IRQHandler	EXTI 线[9:5] 中断
TIMER0_BRK_TIMER8_IRQHandler	TIMER0 中止中断和 TIMER8 全局中断
TIMER0_UP_TIMER9_IRQHandler	TIMER0 更新中断和 TIMER9 全局中断
TIMER0_TRG_CMT_TIMER10_IRQHandler	TIMER0 触发与通道换相中断和 TIMER10 全局中断
TIMER0_Channel_IRQHandler	TIMER0 通道捕获比较中断
TIMER1_IRQHandler	TIMER1 全局中断
TIMER2_IRQHandler	TIMER2 全局中断
TIMER3_IRQHandler	TIMER3 全局中断
I2C0_EV_IRQHandler	I2C0 事件中断

I2C0_ER_IRQn	I2C0 错误中断
I2C1_EV_IRQn	I2C1 事件中断
I2C1_ER_IRQn	I2C1 错误中断
SPI0_IRQn	SPI0 全局中断
SPI1_IRQn	SPI1 全局中断
USART0_IRQn	USART0 全局中断
USART1_IRQn	USART1 全局中断
USART2_IRQn	USART2 全局中断
EXTI10_15_IRQn	EXTI 线[15:10] 中断
RTC_ALARM_IRQn	连接 EXTI 线的 RTC 闹钟中断
USBD_WKUP_IRQn /	连接 EXTI 线的 USBFS 唤醒中断
USBFS_WKUP_IRQn	USBFS 唤醒中断
TIMER7_BRK_TIMER11_IRQn	TIMER7 中止中断和 TIMER11 全局中断
TIMER7_UP_TIMER12_IRQn	TIMER7 更新中断和 TIMER12 全局中断
TIMER7_TRG_CMT_TIMER13_IRQn	TIMER7 触发与通道换相中断和 TIMER13 全局中断
TIMER7_Channel_IRQn	TIMER7 通道捕获比较中断
EXMC_IRQn	EXMC 全局中断
SDIO_IRQn	SDIO 全局中断
TIMER4_IRQn	TIMER4 全局中断
SPI2_IRQn	SPI2 全局中断
UART3_IRQn	UART3 全局中断
UART4_IRQn	UART4 全局中断
TIMER5_IRQn	TIMER5 全局中断
TIMER6_IRQn	TIMER6 全局中断
DMA1_Channel0_IRQn	DMA1 通道 0 全局中断
DMA1_Channel1_IRQn	DMA1 通道 1 全局中断
DMA1_Channel2_IRQn	DMA1 通道 2 全局中断
DMA1_Channel3_Channel4_IRQn / DMA1_Channel3_IRQn	DMA1 通道 3 全局中断和 DMA1 通道 4 全局中断 / DMA1 通道 3 全局中断
DMA1_Channel4_IRQn	DMA1 通道 4 全局中断
ENET_IRQn	ENET 中断
ENET_WKUP_IRQn	ENET 唤醒中断
CAN1_TX_IRQn	CAN1 发送中断
CAN1_RX0_IRQn	CAN1 接收 0 中断
CAN1_RX1_IRQn	CAN1 接收 1 中断
CAN1_EWMC_IRQn	CAN1 EWMC 中断
USBFS_IRQn	USBFS 全局中断

MISC库函数列表如下表所示：

表 3-446. MISC 库函数

库函数名称	库函数描述
nvic_priority_group_set	设置优先级组
nvic_irq_enable	使能NVIC的中断
nvic_irq_disable	禁能NVIC的中断
nvic_vector_table_set	设置向量表地址
system_lowpower_set	设置系统低功耗模式状态
system_lowpower_reset	复位系统低功耗模式状态
systick_clksource_set	设置系统定时器时钟源

函数 nvic_priority_group_set

函数nvic_priority_group_set描述见下表:

表 3-447. 函数 nvic_priority_group_set

函数名称	nvic_priority_group_set
函数原形	void nvic_priority_group_set(uint32_t nvic_prigroup);
功能描述	配置优先级组的位长度
先决条件	-
被调用函数	-
输入参数{in}	
nvic_prigroup	优先级组
NVIC_PRIGROUP_PRE0_SUB4	0位用于抢占优先级, 4位用于响应优先级
NVIC_PRIGROUP_PRE1_SUB3	1位用于抢占优先级, 3位用于响应优先级
NVIC_PRIGROUP_PRE2_SUB2	2位用于抢占优先级, 2位用于响应优先级
NVIC_PRIGROUP_PRE3_SUB1	3位用于抢占优先级, 1位用于响应优先级
NVIC_PRIGROUP_PRE4_SUB0	4位用于抢占优先级, 0位用于响应优先级
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* priority group configuration , 0 bits for pre-emption priority 4 bits for subpriority */
nvic_priority_group_set(NVIC_PRIGROUP_PRE0_SUB4);
```

函数 nvic_irq_enable

函数nvic_irq_enable描述见下表：

表 3-448. 函数 nvic_irq_enable

函数名称	nvic_irq_enable
函数原形	void nvic_irq_enable(uint8_t nvic_irq, uint8_t nvic_irq_pre_priority, uint8_t nvic_irq_sub_priority);
功能描述	使能中断，配置中断的优先级
先决条件	-
被调用函数	nvic_priority_group_set
输入参数{in}	
nvic_irq	NVIC中断，参考枚举类型 表 3-445. 枚举类型IRQn_Type
输入参数{in}	
nvic_irq_pre_priority	抢占优先级（0~4）
输入参数{in}	
nvic_irq_sub_priority	响应优先级（0~4）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable window watchDog timer interrupt , pre-emption priority is 1, subpriority is 1 */
```

```
nvic_irq_enable(WWDGT_IRQn, 1, 1);
```

函数 nvic_irq_disable

函数nvic_irq_disable描述见下表:

表 3-449. 函数 nvic_irq_disable

函数名称	nvic_irq_disable	
函数原形	void nvic_irq_disable (uint8_t nvic_irq);	
功能描述	禁能中断	
先决条件	-	
被调用函数	-	
输入参数{in}		
nvic_irq	NVIC中断, 参考枚举类型 表 3-445. 枚举类型IRQn_Type	
输出参数{out}		
-	-	
返回值		
-	-	

例如:

```
/* disable window watchDog timer interrupt */
nvic_irq_disable(WWDGT_IRQn);
```

函数 nvic_vector_table_set

函数nvic_vector_table_set描述见下表:

表 3-450. 函数 nvic_vector_table_set

函数名称	nvic_vector_table_set	
函数原形	void nvic_vector_table_set(uint32_t nvic_vict_tab, uint32_t offset);	
功能描述	设置向量表地址	
先决条件	-	
被调用函数	-	
输入参数{in}		
nvic_vict_tab	RAM 或者 FLASH基址	

<i>NVIC_VECTTAB_RAM</i>	RAM 基地址
<i>NVIC_VECTTAB_FLASH</i>	FLASH基地址
输入参数{in}	
offset	向量表偏移量（向量表地址=基地址+偏移量）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set vector table address = NVIC_VECTTAB_FLASH +0x200 */
nvic_vector_table_set (NVIC_VECTTAB_FLASH,0x200);
```

函数 system_lowpower_set

函数system_lowpower_set描述见下表：

表 3-451. 函数 system_lowpower_set

函数名称	system_lowpower_set
函数原形	void system_lowpower_set(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
<i>SCB_LPM_SLEEP_EXIT_ISR</i>	该位为1时，退出ISR时一直处于低功耗模式
<i>SCB_LPM_DEEPSLEEP</i>	该位为1时，系统处于deep sleep模式
<i>SCB_LPM_WAKE_BY_ALL_INT</i>	该位为1时，低功耗模式可以被所有中断唤醒（无论中断是否被使能）
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* the system always enter low power mode by exiting from ISR */
system_lowpower_set(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 **system_lowpower_reset**

函数**system_lowpower_reset**描述见下表：

表 3-452. 函数 **system_lowpower_reset**

函数名称	system_lowpower_reset
函数原形	void system_lowpower_reset(uint8_t lowpower_mode);
功能描述	系统低功耗模式状态的管理
先决条件	-
被调用函数	-
输入参数{in}	
lowpower_mode	系统低功耗模式的状态
SCB_LPM_SLEEP_EXIT_ISR	该位为0时，退出ISR时退出低功耗模式
SCB_LPM_DEEPSLEEP	该位为0时，系统进入sleep模式
SCB_LPM_WAKE_BY_ALL_INT	该位为0时，系统只能被使能的中断唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* the system will exit low power mode by exiting from ISR */
system_lowpower_reset(SCB_LPM_SLEEP_EXIT_ISR);
```

函数 systick_clksource_set

函数systick_clksource_set描述见下表:

表 3-453. 函数 systick_clksource_set

函数名称	systick_clksource_set
函数原形	void systick_clksource_set(uint32_t systick_clksource);
功能描述	设置SysTick时钟源
先决条件	-
被调用函数	-
输入参数{in}	
systick_clksource	SysTick时钟源
SYSTICK_CLKSOU RCE_HCLK	SysTick时钟源为AHB时钟
SYSTICK_CLKSOU RCE_HCLK_DIV8	SysTick时钟源为AHB时钟的8分频
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* systick clock source is HCLK/8 */  
  
systick_clksource_set (SYSTICK_CLKSOURCE_HCLK_DIV8);
```

3.18. PMU

电源管理单元提供了三种省电模式，包括睡眠模式，深度睡眠模式和待机模式。章节 [3.18.1](#) 描述了 PMU 的寄存器列表，章节 [3.18.2](#) 对 PMU 库函数进行说明。

3.18.1. 外设寄存器说明

PMU 寄存器列表如下表所示:

表 3-454. PMU 寄存器

寄存器名称	寄存器描述
PMU_CTL	控制寄存器
PMU_CS	电源控制和状态寄存器

3.18.2. 外设库函数说明

PMU 库函数列表如下表所示：

表 3-455. PMU 库函数

库函数名称	库函数描述
pmu_deinit	复位外设PMU
pmu_lvd_select	选择低压检测阈值
pmu_ldo_output_select	LDO输出电压选择
pmu_lvd_disable	关闭低压检测器
pmu_highdriver_switch_select	高驱动模式切换器选择
pmu_highdriver_mode_enable	使能高驱动模式
pmu_highdriver_mode_disable	失能高驱动模式
pmu_lowdriver_mode_enable	深度睡眠模式下使能低驱动模式
pmu_lowdriver_mode_disable	深度睡眠模式下失能低驱动模式
pmu_lowpower_driver_config	使用低功耗LDO时驱动模式选择
pmu_normalpower_driver_config	使用正常LDO时驱动模式选择
pmu_to_sleepmode	进入睡眠模式
pmu_to_deepsleepmode	进入深度睡眠模式
pmu_to_standbymode	进入待机模式
pmu_wakeup_pin_enable	WKUP引脚唤醒使能
pmu_wakeup_pin_disable	WKUP引脚唤醒失能
pmu_backup_write_enable	备份域写使能
pmu_backup_write_disable	备份域写失能
pmu_flag_clear	清除标志位
pmu_flag_get	获取标志位

函数 pmu_deinit

函数pmu_deinit描述见下表:

表 3-456. 函数 pmu_deinit

函数名称	pmu_deinit
函数原型	void pmu_deinit(void);
功能描述	复位外设PMU
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset PMU */
pmu_deinit();
```

函数 pmu_lvd_select

函数pmu_lvd_select描述见下表:

表 3-457. 函数 pmu_lvd_select

函数名称	pmu_lvd_select
函数原型	void pmu_lvd_select(uint32_t lvdt_n);
功能描述	选择低压检测阈值
先决条件	-
被调用函数	-
输入参数{in}	
lvdt_n	电压阈值
PMU_LVDT_0	电压阈值为2.2V

<i>PMU_LVDT_1</i>	电压阈值为2.3V
<i>PMU_LVDT_2</i>	电压阈值为2.4V
<i>PMU_LVDT_3</i>	电压阈值为2.5V
<i>PMU_LVDT_4</i>	电压阈值为2.6V
<i>PMU_LVDT_5</i>	电压阈值为2.7V
<i>PMU_LVDT_6</i>	电压阈值为2.8V
<i>PMU_LVDT_7</i>	电压阈值为2.9V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select low voltage detector threshold as 2.9V */
pmu_lvd_select (PMU_LVDT_7);
```

函数 **pmu_ldo_output_select**

函数pmu_ldo_output_select描述见下表：

表 3-458. 函数 **pmu_ldo_output_select**

函数名称	pmu_ldo_output_select
函数原型	void pmu_ldo_output_select(uint32_t ldo_output);
功能描述	内部电压调节器（LDO）输出电压选择
先决条件	-
被调用函数	-
输入参数{in}	
ldo_output	输出电压模式
<i>PMU_LDOVS_LOW</i>	输出低电压模式
<i>PMU_LDOVS_MID</i>	输出中电压模式
<i>PMU_LDOVS_HIGH</i>	输出高电压模式

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select output low voltage mode */
pmu_ldo_output_select (PMU_LDOVS_LOW);
```

函数 pmu_lvd_disable

函数pmu_lvd_disable描述见下表：

表 3-459. 函数 pmu_lvd_disable

函数名称	pmu_lvd_disable
函数原型	void pmu_lvd_disable (void);
功能描述	关闭低压检测器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable PMU lvd */
pmu_lvd_disable ();
```

函数 pmu_highdriver_switch_select

函数pmu_highdriver_switch_select描述见下表：

表 3-460. 函数 pmu_highdriver_switch_select

函数名称	pmu_highdriver_switch_select
------	------------------------------

函数原型	void pmu_highdriver_switch_select (uint32_t highdr_switch);
功能描述	高驱动模式切换器选择
先决条件	-
被调用函数	-
输入参数{in}	
highdr_switch	使能或失能高驱动模式切换器
PMU_HIGHDR_SWITCH_NONE	失能高驱动模式切换器
PMU_HIGHDR_SWITCH_EN	使能高驱动模式切换器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable high-driver mode switch */

pmu_highdriver_switch_select (PMU_HIGHDR_SWITCH_EN);
```

函数 pmu_highdriver_mode_enable

函数pmu_highdriver_mode_enable描述见下表：

表 3-461. 函数 pmu_highdriver_mode_enable

函数名称	pmu_highdriver_mode_enable
函数原型	void pmu_highdriver_mode_enable(void);
功能描述	使能高驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* enable high-driver mode */
pmu_highdriver_mode_enable();
```

函数 pmu_highdriver_mode_disable

函数pmu_highdriver_mode_disable描述见下表：

表 3-462. 函数 pmu_highdriver_mode_disable

函数名称	pmu_highdriver_mode_disable
函数原型	void pmu_highdriver_mode_disable(void);
功能描述	失能高驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable high-driver mode */
pmu_highdriver_mode_disable();
```

函数 pmu_lowdriver_mode_enable

函数pmu_lowdriver_mode_enable描述见下表：

表 3-463. 函数 pmu_lowdriver_mode_enable

函数名称	pmu_lowdriver_mode_enable
函数原型	void pmu_lowdriver_mode_enable (void);
功能描述	使能低驱动模式

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable low-driver mode */
pmu_lowdriver_mode_enable();
```

函数 **pmu_lowdriver_mode_disable**

函数pmu_lowdriver_mode_disable描述见下表：

表 3-464. 函数 **pmu_lowdriver_mode_disable**

函数名称	pmu_lowdriver_mode_disable
函数原型	void pmu_lowdriver_mode_disable (void);
功能描述	失能低驱动模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable low-driver mode */
pmu_lowdriver_mode_disable();
```

函数 pmu_lowpower_driver_config

函数pmu_lowpower_driver_config描述见下表:

表 3-465. 函数 pmu_lowpower_driver_config

函数名称	pmu_lowpower_driver_config
函数原型	void pmu_lowpower_driver_config (void);
功能描述	低功耗LDO下驱动模式配置
先决条件	-
被调用函数	-
输入参数{in}	
mode	驱动模式
PMU_NORMALDR_LOWPWR	正常驱动模式
PMU_LOWDR_LOWPWR	低驱动模式
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* low-driver mode when use low power LDO */
pmu_lowpower_driver_config (PMU_NORMALDR_LOWPWR);
```

函数 pmu_normalpower_driver_config

函数pmu_normalpower_driver_config描述见下表:

表 3-466. 函数 pmu_normalpower_driver_config

函数名称	pmu_normalpower_driver_config
函数原型	void pmu_normalpower_driver_config (void);
功能描述	正常功耗LDO下驱动模式配置
先决条件	-
被调用函数	-

输入参数{in}	
mode	驱动模式
PMU_NORMALDR_LOWPWR	正常驱动模式
PMU_LOWDR_NORMAPWR	低驱动模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* normal driver when use low power LDO */
pmu_normalpower_driver_config (PMU_LOWDR_NORMAPWR);
```

函数 pmu_to_sleepmode

函数pmu_to_sleepmode描述见下表：

表 3-467. 函数 pmu_to_sleepmode

函数名称	pmu_to_sleepmode
函数原型	void pmu_to_sleepmode(uint8_t sleepmodecmd);
功能描述	进入睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
sleepmodecmd	进入睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at sleep mode */

pmu_to_sleepmode (WFI_CMD);
```

函数 pmu_to_deepsleepmode

函数pmu_to_deepsleepmode描述见下表：

表 3-468. 函数 pmu_to_deepsleepmode

函数名称	pmu_to_deepsleepmode
函数原型	void pmu_to_deepsleepmode(uint32_t ldo,uint8_t deepsleepmodecmd);
功能描述	进入深度睡眠模式
先决条件	-
被调用函数	-
输入参数{in}	
ldo	LDO工作模式
PMU_LDO_NORMAL	当系统进入深度睡眠模式时，LDO仍正常工作
PMU_LDO_LOWPOWER	当系统进入深度睡眠模式时，LDO进入低功耗模式
输入参数{in}	
deepsleepmodecmd	进入深度睡眠模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* PMU work at deepsleep mode */

pmu_to_deepsleepmode (PMU_LDO_NORMAL, WFI_CMD);
```

函数 pmu_to_standbymode

函数pmu_to_standbymode描述见下表:

表 3-469. 函数 pmu_to_standbymode

函数名称	pmu_to_standbymode
函数原型	void pmu_to_standbymode(uint8_t standbymodecmd);
功能描述	进入待机模式
先决条件	-
被调用函数	-
输入参数{in}	
standbymodecmd	进入待机模式命令
WFI_CMD	WFI命令
WFE_CMD	WFE命令
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* PMU work at standby mode */

pmu_to_standby (WFI_CMD);
```

函数 pmu_backup_write_enable

函数pmu_backup_write_enable描述见下表:

表 3-470. 函数 pmu_backup_write_enable

函数名称	pmu_backup_write_enable
函数原型	void pmu_backup_write_enable (void);
功能描述	备份域写使能
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable backup domain write */

pmu_backup_write_enable();
```

函数 pmu_backup_write_disable

函数pmu_backup_write_disable描述见下表：

表 3-471. 函数 pmu_backup_write_disable

函数名称	pmu_backup_write_disable
函数原型	void pmu_backup_write_disable (void);
功能描述	备份域写失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable backup domain write */

pmu_backup_write_disable();
```

函数 pmu_wakeup_pin_enable

函数pmu_wakeup_pin_enable描述见下表：

表 3-472. 函数 pmu_wakeup_pin_enable

函数名称	pmu_wakeup_pin_enable
函数原型	void pmu_wakeup_pin_enable(void);
功能描述	WKUP引脚唤醒使能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable wakeup pin */
pmu_wakeup_pin_enable();
```

函数 pmu_wakeup_pin_disable

函数pmu_wakeup_pin_disable描述见下表：

表 3-473. 函数 pmu_wakeup_pin_disable

函数名称	pmu_wakeup_pin_disable
函数原型	void pmu_wakeup_pin_disable (void);
功能描述	WKUP引脚唤醒失能
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable wakeup pin */
pmu_wakeup_pin_disable();
```

函数 pmu_flag_get

函数pmu_flag_get描述见下表：

表 3-474. 函数 pmu_flag_get

函数名称	pmu_flag_get
函数原型	FlagStatus pmu_flag_get(uint32_t flag_reset);
功能描述	获取标志位
先决条件	-
被调用函数	-
输入参数{in}	
<i>flag</i>	标志位
<i>PMU_FLAG_RESET_WAKEUP</i>	唤醒标志
<i>PMU_FLAG_RESET_STANDBY</i>	待机标志
<i>PMU_FLAG_LVD</i>	低电压检测标志
<i>PMU_FLAG_LDOV_SRF</i>	LDO电压选择就绪标志
<i>PMU_FLAG_HDRF</i>	高驱动就绪标志
<i>PMU_FLAG_HDSRF</i>	高驱动切换器就绪标志
<i>PMU_FLAG_LDRF</i>	低驱动就绪标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag state */

FlagStatus status;

status = pmu_flag_get (PMU_FLAG_RESET_STANDBY);
```

函数 pmu_flag_clear

函数pmu_flag_clear描述见下表：

表 3-475. 函数 pmu_flag_clear

函数名称	pmu_flag_clear
函数原型	void pmu_flag_clear(uint32_t flag_reset);
功能描述	清除标志位
先决条件	-
被调用函数	-
输入参数{in}	
flag_reset	标志位
PMU_FLAG_RESET_WAKEUP	清除唤醒标志
PMU_FLAG_RESET_STANDBY	清除待机标志
输出参数{out}	
-	
返回值	
-	

例如：

```
/* clear flag bit */

pmu_flag_clear (PMU_FLAG_RESET_WAKEUP);
```

3.19. RCU

RCU 是复位和时钟单元，复位控制包括三种控制方式：电源复位、系统复位和备份域复位。时钟控制单元提供了一系列频率的时钟功能。章节 [3.19.1](#) 描述了 RCU 的寄存器列表，章节 [3.19.2](#) 对 RCU 库函数进行说明。

3.19.1. 外设寄存器说明

RCU寄存器列表如下表所示：

表 3-476. RCU 寄存器

寄存器名称	寄存器描述
RCU_CTL	控制寄存器
RCU_CFG0	时钟配置寄存器0
RCU_INT	时钟中断寄存器
RCU_APB2RST	APB2复位寄存器
RCU_APB1RST	APB1复位寄存器
RCU_AHBEN	AHB使能寄存器
RCU_APB2EN	APB2使能寄存器
RCU_APB1EN	APB1使能寄存器
RCU_BDCTL	备份域控制寄存器
RCU_RSTSCK	复位源/时钟寄存器
RCU_AHBRST	AHB复位寄存器(CL型)
RCU_CFG1	时钟配置寄存器1
RCU_DSV	深度睡眠模式电压寄存器
RCU_ADDCTL	附加时钟控制寄存器
RCU_ADDINT	附加时钟中断寄存器
RCU_ADDAPB1RS _T	APB1附加复位寄存器
RCU_ADDAPB1EN	APB1附加使能寄存器

3.19.2. 外设库函数说明

RCU库函数列表如下表所示：

表 3-477. RCU 库函数

库函数名称	库函数描述
rcu_deinit	复位RCU

库函数名称	库函数描述
rcu_periph_clock_enable	使能外设时钟
rcu_periph_clock_disable	禁能外设时钟
rcu_periph_clock_sleep_enable	在睡眠模式下，使能外设时钟
rcu_periph_clock_sleep_disable	在睡眠模式下，禁能外设时钟
rcu_periph_reset_enable	使能外设复位
rcu_periph_reset_disable	禁能外设复位
rcu_bkp_reset_enable	使能BKP复位
rcu_bkp_reset_disable	禁能BKP复位
rcu_system_clock_source_config	配置选择系统时钟源
rcu_system_clock_source_get	获取系统时钟源选择状态
rcu_ahb_clock_config	配置AHB时钟预分频选择
rcu_apb1_clock_config	配置APB1时钟预分频选择
rcu_apb2_clock_config	配置APB2时钟预分频选择
rcu_ckout0_config	配置CKOUT0时钟源选择
rcu_pll_config	配置主PLL时钟
rcu_pllpresel_config	配置PLL时钟源选择
rcu_predv0_config	配置PREDV0分频因子
rcu_predv1_config	配置PREDV1分频因子
rcu_pll1_config	配置PLL1时钟
rcu_pll2_config	配置PLL2时钟
rcu_adc_clock_config	配置ADC的时钟分频系数
rcu_usb_clock_config	配置USB的时钟分频系数
rcu_rtc_clock_config	配置RTC的时钟源选择
rcu_i2s1_clock_config	配置I2S1的时钟源选择
rcu_i2s2_clock_config	配置I2S2的时钟源选择
rcu_ck48m_clock_config	配置CK48M时钟选择
rcu_flag_get	获取时钟稳定和外设复位标志

库函数名称	库函数描述
rcu_all_reset_flag_clear	清除所有复位标志位
rcu_interrupt_flag_get	获取时钟稳定中断和时钟阻塞中断标志
rcu_interrupt_flag_clear	清除中断标志
rcu_interrupt_enable	使能时钟稳定中断
rcu_interrupt_disable	禁能时钟稳定中断
rcu_lxtal_drive_capability_config	配置LXTAL驱动能力
rcu_osc_stab_wait	等待振荡器稳定标志位置位或振荡器起振超时
rcu_osc_on	打开振荡器
rcu_osc_off	关闭振荡器
rcu_osc_bypass_mode_enable	使能振荡器时钟旁路模式
rcu_osc_bypass_mode_disable	禁能振荡器时钟旁路模式
rcu_hxtal_clock_monitor_enable	使能HXTAL时钟监视器
rcu_hxtal_clock_monitor_disable	禁能HXTAL时钟监视器
rcu_irc8m_adjust_value_set	设置内部8MHz RC振荡器时钟调整值
rcu_deepsleep_voltage_set	设置深度睡眠模式电压值
rcu_clock_freq_get	获取系统时钟、总线频率

函数 `rcu_deinit`

函数`rcu_deinit`描述见下表:

表 3-478. 函数 `rcu_deinit`

函数名称	rcu_deinit
函数原形	void rcu_deinit(void);
功能描述	复位RCU，将RCU所有寄存器的值复位成初始值
先决条件	-
被调用函数	rcu_osc_stab_wait
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* deinitialize the RCU */
rcu_deinit();
```

函数 **rcu_periph_clock_enable**

函数rcu_periph_clock_enable描述见下表：

表 3-479. 函数 **rcu_periph_clock_enable**

函数名称	rcu_periph_clock_enable
函数原形	void rcu_periph_clock_enable(rcu_periph_enum periph);
功能描述	使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，具体参考rcu_periph_enum
<i>RCU_GPIOx</i>	GPIOx时钟(x=A,B,C,D,E,F,G)
<i>RCU_AF</i>	复用功能时钟
<i>RCU_CRC</i>	CRC时钟
<i>RCU_DMAx</i>	DMAx时钟(x=0,1)
<i>RCU_ENET</i>	ENET时钟(CL型)
<i>RCU_ENETTX</i>	ENETTX时钟(CL型)
<i>RCU_ENETRX</i>	ENETRX时钟(CL型)
<i>RCU_USBD</i>	USBD时钟(HD、XD型)
<i>RCU_USBFS</i>	USBFS时钟(CL型)
<i>RCU_EXMC</i>	EXMC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWWDGT时钟

<i>RCU_SPIx</i>	SPIx时钟(x=0,1,2)
<i>RCU_USARTTx</i>	USARTx时钟(x=0,1,2)
<i>RCU_UARTx</i>	UARTx时钟(x=3,4)
<i>RCU_I2Cx</i>	I2Cx时钟(x=0,1)
<i>RCU_CANx</i>	CANx时钟(x=0,1) (仅CL型有CAN1)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟
<i>RCU_RTC</i>	RTC时钟
<i>RCU_ADCx</i>	ADCx时钟(x=0,1,2) (CL型无ADC2)
<i>RCU_SDIO</i>	SDIO时钟 (HD、XD型)
<i>RCU_CTC</i>	CTC时钟
<i>RCU_BKPI</i>	BKP接口时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the USART0 clock */
rcu_periph_clock_enable(RCU_USART0);
```

函数 **rcu_periph_clock_disable**

函数**rcu_periph_clock_disable**描述见下表：

表 3-480. 函数 **rcu_periph_clock_disable**

函数名称	rcu_periph_clock_disable
函数原形	void rcu_periph_clock_disable(rcu_periph_enum periph);
功能描述	禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	

periph	RCU外设，具体参考rcu_periph_enum
<i>RCU_GPIOx</i>	GPIOx时钟(x=A,B,C,D,E,F,G)
<i>RCU_AF</i>	复用功能时钟
<i>RCU_CRC</i>	CRC时钟
<i>RCU_DMAt</i>	DMAx时钟(x=0,1)
<i>RCU_ENET</i>	ENET时钟 (CL型)
<i>RCU_ENETTX</i>	ENETTX时钟 (CL型)
<i>RCU_ENETRX</i>	ENETRX时钟 (CL型)
<i>RCU_USBD</i>	USBD时钟 (HD、XD型)
<i>RCU_USBFS</i>	USBFS时钟 (CL型)
<i>RCU_EXMC</i>	EXMC时钟
<i>RCU_TIMERx</i>	TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGT</i>	WWDGT时钟
<i>RCU_SPIx</i>	SPIx时钟(x=0,1,2)
<i>RCU_USARTx</i>	USARTx时钟(x=0,1,2)
<i>RCU_UARTx</i>	UARTx时钟(x=3,4)
<i>RCU_I2Cx</i>	I2Cx时钟(x=0,1)
<i>RCU_CANx</i>	CANx时钟(x=0,1) (仅CL型有CAN1)
<i>RCU_PMU</i>	PMU时钟
<i>RCU_DAC</i>	DAC时钟
<i>RCU_RTC</i>	RTC时钟
<i>RCU_ADCx</i>	ADCx时钟(x=0,1,2) (CL型无ADC2)
<i>RCU_SDIO</i>	SDIO时钟 (HD、XD型)
<i>RCU_CTC</i>	CTC时钟
<i>RCU_BKPI</i>	BKP接口时钟
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable the USART0 clock */
rcu_periph_clock_disable(RCU_USART0);
```

函数 **rcu_periph_clock_sleep_enable**

函数rcu_periph_clock_sleep_enable描述见下表：

表 3-481. 函数 **rcu_periph_clock_sleep_enable**

函数名称	rcu_periph_clock_sleep_enable
函数原形	void rcu_periph_clock_sleep_enable(rcu_periph_sleep_enum periph);
功能描述	在睡眠模式下，使能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考rcu_periph_sleep_enum
RCU_FMC_SLP	FMC时钟
RCU_SRAM_SLP	SRAM时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_enable(RCU_FMC_SLP);
```

函数 **rcu_periph_clock_sleep_disable**

函数rcu_periph_clock_sleep_disable描述见下表：

表 3-482. 函数 **rcu_periph_clock_sleep_disable**

函数名称	rcu_periph_clock_sleep_disable
函数原形	void rcu_periph_clock_sleep_disable(rcu_periph_sleep_enum periph);

功能描述	在睡眠模式下，禁能外设时钟
先决条件	-
被调用函数	-
输入参数{in}	
periph	RCU外设，参考rcu_periph_sleep_enum
<i>RCU_FMC_SLP</i>	FMC时钟
<i>RCU_SRAM_SLP</i>	SRAM时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the FMC clock when in sleep mode */
rcu_periph_clock_sleep_disable(RCU_FMC_SLP);
```

函数 **rcu_periph_reset_enable**

函数rcu_periph_reset_enable描述见下表：

表 3-483. 函数 **rcu_periph_reset_enable**

函数名称	rcu_periph_reset_enable
函数原形	void rcu_periph_reset_enable(rcu_periph_reset_enum periph_reset);
功能描述	使能外设复位
先决条件	-
被调用函数	-
输入参数{in}	
periph_reset	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,D,E,F,G)
<i>RCU_AFRST</i>	复位复用功能时钟
<i>RCU_ENETRST</i>	复位ENET时钟 (CL型)
<i>RCU_USBDRST</i>	复位USBD时钟 (HD、XD型)

<i>RCU_USBFSRST</i>	复位USBFS时钟 (CL型)
<i>RCU_TIMERxRST</i>	复位TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	复位WWDGTRST
<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1,2)
<i>RCU_USARTTxRST</i>	复位USARTTx时钟(x=0,1,2)
<i>RCU_UARTxRST</i>	复位UARTx时钟(x=3,4)
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1)
<i>RCU_CANxRST</i>	复位CANx时钟(x=0,1) (仅CL型有CAN1)
<i>RCU_PMURST</i>	复位PMU时钟
<i>RCU_DACRST</i>	复位DAC时钟
<i>RCU_ADCxRST</i>	复位ADCx时钟(x=0,1,2) (CL型无ADC2)
<i>RCU_CTCRST</i>	复位CTC
<i>RCU_BKPIRST</i>	复位BKP接口时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 reset */
rcu_periph_reset_enable(RCU_SPI0RST);
```

函数 **rcu_periph_reset_disable**

函数rcu_periph_reset_disable描述见下表：

表 3-484. 函数 **rcu_periph_reset_disable**

函数名称	rcu_periph_reset_disable
函数原形	void rcu_periph_reset_disable(rcu_periph_reset_enum periph_reset);
功能描述	禁能外设复位
先决条件	-
被调用函数	-

输入参数{in}	
periph_reset	RCU外设复位，参考rcu_periph_reset_enum
<i>RCU_GPIOxRST</i>	复位GPIOx时钟(x=A,B,C,D,E,F,G)
<i>RCU_AFRST</i>	复位复用功能时钟
<i>RCU_ENETRST</i>	复位ENET时钟（CL型）
<i>RCU_USBDRST</i>	复位USBD时钟（HD、XD型）
<i>RCU_USBFSRST</i>	复位USBFS时钟（CL型）
<i>RCU_TIMERxRST</i>	复位TIMERx时钟(x=0,1,2,3,4,5,6,7,8,9,10,11,12,13)
<i>RCU_WWDGTRST</i>	复位WWDGTRST时钟
<i>RCU_SPIxRST</i>	复位SPIx时钟(x=0,1,2)
<i>RCU_USARTxRST</i>	复位USARTx时钟(x=0,1,2)
<i>RCU_UARTxRST</i>	复位UARTx时钟(x=3,4)
<i>RCU_I2CxRST</i>	复位I2Cx时钟(x=0,1)
<i>RCU_CANxRST</i>	复位CANx时钟(x=0,1) (仅CL型有CAN1)
<i>RCU_PMURST</i>	复位PMU时钟
<i>RCU_DACRST</i>	复位DAC时钟
<i>RCU_ADCxRST</i>	复位ADCx时钟(x=0,1,2) (CL型无ADC2)
<i>RCU_CTCRST</i>	复位CTC时钟
<i>RCU_BKPIRST</i>	复位BKP接口时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 reset */
rcu_periph_reset_disable(RCU_SPI0RST);
```

函数 **rcu_bkp_reset_enable**

函数rcu_bkp_reset_enable描述见下表：

表 3-485. 函数 `rcu_bkp_reset_enable`

函数名称	<code>rcu_bkp_reset_enable</code>
函数原形	<code>void rcu_bkp_reset_enable(void);</code>
功能描述	使能BKP复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* reset the BKP domain */

rcu_bkp_reset_enable();
```

函数 `rcu_bkp_reset_disable`

函数 `rcu_bkp_reset_disable` 描述见下表：

表 3-486. 函数 `rcu_bkp_reset_disable`

函数名称	<code>rcu_bkp_reset_disable</code>
函数原形	<code>void rcu_bkp_reset_disable(void);</code>
功能描述	禁能BKP复位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable the BKP domain reset */
```

```
rcu_bkp_reset_disable();
```

函数 **rcu_system_clock_source_config**

函数rcu_system_clock_source_config描述见下表：

表 3-487. 函数 **rcu_system_clock_source_config**

函数名称	rcu_system_clock_source_config
函数原形	void rcu_system_clock_source_config(uint32_t ck_sys);
功能描述	配置选择系统时钟源
先决条件	-
被调用函数	-
输入参数{in}	
ck_sys	系统时钟源选择
<i>RCU_CKSYSRC_I</i> <i>RC8M</i>	选择CK_IRC8M时钟作为CK_SYS时钟源
<i>RCU_CKSYSRC_HXTAL</i>	选择CK_HXTAL时钟作为CK_SYS时钟源
<i>RCU_CKSYSRC_PLL</i>	选择CK_PLL时钟作为CK_SYS时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the CK_HXTAL as the CK_SYS source */
```

```
rcu_system_clock_source_config(RCU_CKSYSRC_HXTAL);
```

函数 **rcu_system_clock_source_get**

函数rcu_system_clock_source_get描述见下表：

表 3-488. 函数 rcu_system_clock_source_get

函数名称	rcu_system_clock_source_get
函数原形	uint32_t rcu_system_clock_source_get(void);
功能描述	获取系统时钟源选择状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	RCU_SCSS_IRC8M/RCU_SCSS_HXTAL/RCU_SCSS_PLL

例如：

```
uint32_t temp_cksys_status;
/* get the CK_SYS source */
temp_cksys_status = rcu_system_clock_source_get();
```

函数 rcu_ahb_clock_config

函数rcu_ahb_clock_config描述见下表：

表 3-489. 函数 rcu_ahb_clock_config

函数名称	rcu_ahb_clock_config
函数原形	void rcu_ahb_clock_config(uint32_t ck_ahb);
功能描述	配置AHB时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_ahb	AHB预分频选择
RCU_AHB_CKSYS_DIVx	选择CK_SYS时钟x分频 (x=1, 2, 4, 8, 16, 64, 128, 256, 512)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure CK_SYS/128 */
rcu_ahb_clock_config(RCU_AHB_CKSYS_DIV128);
```

函数 rcu_apb1_clock_config

函数rcu_apb1_clock_config描述见下表：

表 3-490. 函数 rcu_apb1_clock_config

函数名称	rcu_apb1_clock_config
函数原形	void rcu_apb1_clock_config(uint32_t ck_apb1);
功能描述	配置APB1时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
ck_apb1	APB1预分频选择
RCU_APB1_CKAH_B_DIV1	选择CK_AHB为CK_APB1
RCU_APB1_CKAH_B_DIV2	选择CK_AHB/2为CK_APB1
RCU_APB1_CKAH_B_DIV4	选择CK_AHB/4为CK_APB1
RCU_APB1_CKAH_B_DIV8	选择CK_AHB/8为CK_APB1
RCU_APB1_CKAH_B_DIV16	选择CK_AHB/16为CK_APB1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/16 as CK_APB1 */
rcu_apb1_clock_config(RCU_APB1_CKAHB_DIV16);
```

函数 `rcu_apb2_clock_config`

函数`rcu_apb2_clock_config`描述见下表：

表 3-491. 函数 `rcu_apb2_clock_config`

函数名称	<code>rcu_apb2_clock_config</code>
函数原形	<code>void rcu_apb2_clock_config(uint32_t ck_apb2);</code>
功能描述	配置APB2时钟预分频选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>ck_apb2</code>	APB2预分频选择
<code>RCU_APB2_CKAH_B_DIV1</code>	选择CK_AHB为CK_APB2
<code>RCU_APB2_CKAH_B_DIV2</code>	选择CK_AHB/2为CK_APB2
<code>RCU_APB2_CKAH_B_DIV4</code>	选择CK_AHB/4为CK_APB2
<code>RCU_APB2_CKAH_B_DIV8</code>	选择CK_AHB/8为CK_APB2
<code>RCU_APB2_CKAH_B_DIV16</code>	选择CK_AHB/16为CK_APB2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure CK_AHB/8 as CK_APB2 */
rcu_apb2_clock_config(RCU_APB2_CKAHB_DIV8);
```

函数 rcu_ckout0_config

函数rcu_ckout0_config描述见下表：

表 3-492. 函数 rcu_ckout0_config

函数名称	rcu_ckout0_config
函数原形	void rcu_ckout0_config(uint32_t ckout0_src);
功能描述	配置CKOUT0时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
ckout0_src	CKOUT0时钟源选择
RCU_CKOUT0SRC_NONE	无时钟输出
RCU_CKOUT0SRC_CKSYS	选择系统时钟CK_SYS
RCU_CKOUT0SRC_IRC8M	选择内部8M RC振荡器时钟
RCU_CKOUT0SRC_HXTAL	选择高速晶体振荡器时钟 (HXTAL)
RCU_CKOUT0SRC_CKPLL_DIV2	选择 (CK_PLL / 2) 时钟
RCU_CKOUT0SRC_CKPLL1	选择CK_PLL1时钟
RCU_CKOUT0SRC_CKPLL2_DIV2	选择 (CK_PLL2 / 2) 时钟
RCU_CKOUT0SRC_CKPLL2	选择CK_PLL2时钟
RCU_CKOUT0SRC_EXT1	选择EXT1时钟
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the HXTAL as CK_OUT0 clock source */

rcu_ckout0_config(RCU_CKOUT0SRC_HXTAL);
```

函数 rcu_pll_config

函数rcu_pll_config描述见下表：

表 3-493. 函数 rcu_pll_config

函数名称	rcu_pll_config
函数原形	void rcu_pll_config(uint32_t pll_src, uint32_t pll_mul);
功能描述	配置主PLL时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_src	PLL时钟源选择
RCU_PLLSRC_IRC8M_DIV2	(IRC8M / 2)被选择为PLL时钟的时钟源
RCU_PLLSRC_HXTAL_IRC48M	HXTAL时钟或者IRC48M时钟被选择为PLL时钟的时钟源
输入参数{in}	
pll_mul	PLL时钟倍频因子
RCU_PLL_MULx	PLL源时钟 * x (XD 型 x = 2..63, CL 型 x = 2..14, 16..63, 6.5)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL */

rcu_pll_config(RCU_PLLSRC_HXTAL, RCU_PLL_MUL10);
```

函数 rcu_pllpresel_config

函数rcu_pllpresel_config描述见下表：

表 3-494. 函数 rcu_pllpresel_config

函数名称	rcu_pllpresel_config
函数原形	void rcu_pllpresel_config(uint32_t pll_presel);
功能描述	配置PLL时钟源预选
先决条件	-
被调用函数	-
输入参数{in}	
pll_presel	PLL时钟源
RCU_PLLPRESRC_HXTAL	PLL输入为HXTAL
RCU_PLLPRESRC_IRC48M	PLL输入为IRC48M
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL preselection */

rcu_pllpresel_config (RCU_PLLPRESRC_HXTAL);
```

函数 rcu_predv0_config (HD、XD 产品)

函数rcu_predv0_config描述见下表：

表 3-495. 函数 rcu_predv0_config

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(uint32_t predv0_div);
功能描述	配置PREDV0分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv0_div	PREDV0分频因子

<i>RCU_PREDV0_DIV</i> <i>x</i>	PREDV0输入源时钟 <i>x</i> 分频 (<i>x</i> = 1..2)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDV0 division factor */

rcu_predv0_config(RCU_PREDV0_DIV1);
```

函数 **rcu_predv0_config** (CL 型产品)

函数rcu_predv0_config描述见下表：

表 3-496. 函数 **rcu_predv0_config**

函数名称	rcu_predv0_config
函数原形	void rcu_predv0_config(uint32_t predv0_source, uint32_t predv0_div);
功能描述	配置PREDV0分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv0_source	PREDV0输入时钟源
<i>RCU_PREDV0SRC_HXTAL_IRC48M</i>	HXTAL或者IRC48M被选择为PREDV0的时钟源
<i>RCU_PREDV0SRC_CKPLL1</i>	CK_PLL1被选择为PREDV0的时钟源
输入参数{in}	
predv0_div	PREDV0分频因子
<i>RCU_PREDV0_DIV</i> <i>x</i>	PREDV0输入源时钟 <i>x</i> 分频 (<i>x</i> = 1..16)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure the PREDV0 division factor */
rcu_predv0_config(RCU_PREDV0SRC_HXTAL_IRC48M, RCU_PREDV0_DIV4);
```

函数 **rcu_predv1_config** (CL 型产品)

函数rcu_predv1_config描述见下表：

表 3-497. 函数 **rcu_predv1_config**

函数名称	rcu_predv1_config
函数原形	void rcu_predv1_config(uint32_t predv1_div);
功能描述	配置PREDV1分频因子
先决条件	-
被调用函数	-
输入参数{in}	
predv1_div	PREDV1分频因子
RCU_PREDV1_DIV_x	PREDV1输入源时钟x分频 (x = 1..16)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PREDV1 division factor */
rcu_predv1_config(RCU_PREDV1_DIV8);
```

函数 **rcu_pll1_config** (CL 型产品)

函数rcu_pll1_config描述见下表：

表 3-498. 函数 **rcu_pll1_config**

函数名称	rcu_pll1_config
------	-----------------

函数原形	void rcu_pll1_config(uint32_t pll_mul);
功能描述	配置PLL1时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_mul	PLL1时钟倍频因子
RCU_PLL1_MULx	PLL1源时钟*x, (x = 8..14,16,20)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the PLL1 clock */
rcu_pll1_config(RCU_PLL1_MUL8);
```

函数 rcu_pll2_config (CL 型产品)

函数rcu_pll2_config描述见下表：

表 3-499. 函数 rcu_pll2_config

函数名称	rcu_pll2_config
函数原形	void rcu_pll2_config(uint32_t pll_mul);
功能描述	配置PLL2时钟
先决条件	-
被调用函数	-
输入参数{in}	
pll_mul	PLL2时钟倍频因子
RCU_PLL2_MULx	PLL2源时钟*x, (x = 8..14,16,20,18..32,40)
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* configure the PLL2 clock */
rcu_pll2_config(RCU_PLL2_MUL8);
```

函数 rcu_adc_clock_config

函数rcu_adc_clock_config描述见下表：

表 3-500. 函数 rcu_adc_clock_config

函数名称	rcu_adc_clock_config
函数原形	void rcu_adc_clock_config(uint32_t adc_psc);
功能描述	配置ADC的时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
adc_psc	ADC分频因子
<i>RCU_CKADC_CKA_PB2_DIV2</i>	$CK_{ADC} = CK_{APB2} / 2$
<i>RCU_CKADC_CKA_PB2_DIV4</i>	$CK_{ADC} = CK_{APB2} / 4$
<i>RCU_CKADC_CKA_PB2_DIV6</i>	$CK_{ADC} = CK_{APB2} / 6$
<i>RCU_CKADC_CKA_PB2_DIV8</i>	$CK_{ADC} = CK_{APB2} / 8$
<i>RCU_CKADC_CKA_PB2_DIV12</i>	$CK_{ADC} = CK_{APB2} / 12$
<i>RCU_CKADC_CKA_PB2_DIV16</i>	$CK_{ADC} = CK_{APB2} / 16$
<i>RCU_CKADC_CKA_HB_DIV5</i>	$CK_{ADC} = CK_{AHB} / 5$
<i>RCU_CKADC_CKA_HB_DIV6</i>	$CK_{ADC} = CK_{AHB} / 6$
<i>RCU_CKADC_CKA</i>	$CK_{ADC} = CK_{AHB} / 10$

<i>HB_DIV10</i>	
<i>RCU_CKADC_CKA</i> <i>HB_DIV20</i>	$CK_{ADC} = CK_{AHB} / 20$
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the ADC prescaler factor */

rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV8);
```

函数 **rcu_usb_clock_config**

函数rcu_usb_clock_config描述见下表：

表 3-501. 函数 **rcu_usb_clock_config**

函数名称	rcu_usb_clock_config
函数原形	void rcu_usb_clock_config(uint32_t usb_psc);
功能描述	配置USB的时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usb_psc	USB时钟分频系数
<i>RCU_CKUSB_CKP</i> <i>LL_DIV1_5</i>	$CK_{USBD}/USBFS = CK_{PLL} / 1.5$
<i>RCU_CKUSB_CKP</i> <i>LL_DIV1</i>	$CK_{USBD}/USBFS = CK_{PLL} / 1$
<i>RCU_CKUSB_CKP</i> <i>LL_DIV2_5</i>	$CK_{USBD}/USBFS = CK_{PLL} / 2.5$
<i>RCU_CKUSB_CKP</i> <i>LL_DIV2</i>	$CK_{USBD}/USBFS = CK_{PLL} / 2$
<i>RCU_CKUSB_CKP</i> <i>LL_DIV3</i>	$CK_{USBD}/USBFS = CK_{PLL} / 3$

<i>RCU_CKUSB_CKP_LL_DIV3_5</i>	CK_USBD/USBFS = CK_PLL / 3.5
<i>RCU_CKUSB_CKP_LL_DIV4</i>	CK_USBD/USBFS = CK_PLL / 4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USB prescaler factor */

rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
```

函数 `rcu_rtc_clock_config`

函数`rcu_rtc_clock_config`描述见下表：

表 3-502. 函数 `rcu_rtc_clock_config`

函数名称	<code>rcu_rtc_clock_config</code>
函数原形	<code>void rcu_rtc_clock_config(uint32_t rtc_clock_source);</code>
功能描述	配置RTC的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
<code>rtc_clock_source</code>	RTC时钟源选择
<i>RCU_RTCSRC_NO_NE</i>	没有时钟
<i>RCU_RTCSRC_LXTAL</i>	选择CK_LXTAL时钟作为RTC的时钟源
<i>RCU_RTCSRC_IRC40K</i>	选择CK_IRC40K时钟作为RTC的时钟源
<i>RCU_RTCSRC_HXTAL_DIV_128</i>	选择CK_HXTAL / 128时钟作为RTC的时钟源
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure the RTC clock source selection */
rcu_rtc_clock_config(RCU_RTCSRC_IRC40K);
```

函数 **rcu_i2s1_clock_config** (CL 型产品)

函数rcu_i2s1_clock_config描述见下表：

表 3-503. 函数 **rcu_i2s1_clock_config**

函数名称	rcu_i2s1_clock_config
函数原形	void rcu_i2s1_clock_config(uint32_t i2s_clock_source);
功能描述	配置I2S1的时钟源选择
先决条件	-
被调用函数	-
输入参数{in}	
i2s_clock_source	I2S1时钟源选择
RCU_I2S1SRC_CK_SYS	系统时钟被选择为I2S1时钟的时钟源
RCU_I2S1SRC_CK_PLL2_MUL2	(CK_PLL2 * 2) 被选择为I2S1时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the I2S1 clock source selection */
rcu_i2s1_clock_config(RCU_I2S1SRC_CKPLL2_MUL2);
```

函数 `rcu_i2s2_clock_config` (CL 型产品)

函数`rcu_i2s2_clock_config`描述见下表:

表 3-504. 函数 `rcu_i2s2_clock_config`

函数名称	<code>rcu_i2s2_clock_config</code>
函数原形	<code>void rcu_i2s2_clock_config(uint32_t i2s_clock_source);</code>
功能描述	配置I2S2的时钟源选择
先决条件	-
被调用函数	
输入参数{in}	
<code>i2s_clock_source</code>	I2S2时钟源选择
<code>RCU_I2S2SRC_CK_SYS</code>	系统时钟被选择为I2S2时钟的时钟源
<code>RCU_I2S2SRC_CK_PLL2_MUL2</code>	(CK_PLL2 * 2) 被选择为I2S2时钟的时钟源
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the I2S2 clock source selection */
rcu_i2s2_clock_config(RCU_I2S2SRC_CKPLL2_MUL2);
```

函数 `rcu_ck48m_clock_config`

函数`rcu_ck48m_clock_config`描述见下表:

表 3-505. 函数 `rcu_ck48m_clock_config`

函数名称	<code>rcu_ck48m_clock_config</code>
函数原形	<code>void rcu_ck48m_clock_config(uint32_t ck48m_clock_source);</code>
功能描述	配置CK48M时钟源选择
先决条件	-
被调用函数	-

输入参数{in}	
<code>ck48m_clock_sour ce</code>	CK48M时钟源
<code>RCU_CK48MSRC_CKPLL</code>	选择CK_PLL为CK_48M时钟源
<code>RCU_CK48MSRC_I RC48M</code>	选择CK_IRC48M为CK48M时钟源
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET 或 RESET

例如：

```
/* configure the CK48M clock source selection */

rcu_ck48m_clock_config (RCU_CK48MSRC_IRC48M);
```

函数 `rcu_flag_get`

函数`rcu_flag_get`描述见下表：

表 3-506. 函数 `rcu_flag_get`

函数名称	<code>rcu_flag_get</code>
函数原形	<code>FlagStatus rcu_flag_get(rcu_flag_enum flag);</code>
功能描述	获取时钟稳定和外设复位标志
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag</code>	时钟稳定和外设复位标志，参考 <code>rcu_flag_enum</code>
<code>RCU_FLAG_IRC8M_STB</code>	IRC8M稳定标志
<code>RCU_FLAG_HXTAL_STB</code>	HXTAL稳定标志
<code>RCU_FLAG_PLLST_B</code>	PLL稳定标志

<i>RCU_FLAG_PLL1S</i> <i>TB</i>	PLL1稳定标志（CL型产品）
<i>RCU_FLAG_PLL2S</i> <i>TB</i>	PLL2稳定标志（CL型产品）
<i>RCU_FLAG_LXTAL</i> <i>STB</i>	LXTAL稳定标志
<i>RCU_FLAG_IRC40</i> <i>KSTB</i>	IRC40K稳定标志
<i>RCU_FLAG_IRC48</i> <i>MSTB</i>	IRC48M稳定标志
<i>RCU_FLAG_EPRS</i> <i>T</i>	外部引脚复位标志
<i>RCU_FLAG_PORR</i> <i>ST</i>	电源复位标志
<i>RCU_FLAG_SWRS</i> <i>T</i>	软件复位标志
<i>RCU_FLAG_FWDG</i> <i>TRST</i>	独立看门狗定时器复位标志
<i>RCU_FLAG_WWD</i> <i>GTRST</i>	窗口看门狗定时器复位标志
<i>RCU_FLAG_LPRST</i>	low-power复位标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the clock stabilization flag */

if(RESET != rcu_flag_get(RCU_FLAG_LXTALSTB)){
}
```

函数 **rcu_all_reset_flag_clear**

函数**rcu_all_reset_flag_clear**描述见下表：

表 3-507. 函数 rcu_all_reset_flag_clear

函数名称	rcu_all_reset_flag_clear
函数原形	void rcu_all_reset_flag_clear(void);
功能描述	清除所有复位标志位
先决条件	-
被调用函数	
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear all the reset flag */
rcu_all_reset_flag_clear();
```

函数 rcu_interrupt_flag_get

函数rcu_interrupt_flag_get描述见下表：

表 3-508. 函数 rcu_interrupt_flag_get

函数名称	rcu_interrupt_flag_get
函数原形	FlagStatus rcu_interrupt_flag_get(rcu_int_flag_enum int_flag);
功能描述	获取时钟稳定中断和时钟阻塞中断标志
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	中断以及CKM标志，参考rcu_int_flag_enum
RCU_INT_FLAG_IR C40KSTB	IRC40K稳定中断标志
RCU_INT_FLAG_L XTALSTB	LXTAL稳定中断标志

<i>RCU_INT_FLAG_IR</i> <i>C8MSTB</i>	IRC8M稳定中断标志
<i>RCU_INT_FLAG_H</i> <i>XTALSTB</i>	HXTAL稳定中断标志
<i>RCU_INT_FLAG_P</i> <i>LLSTB</i>	PLL稳定中断标志
<i>RCU_INT_FLAG_P</i> <i>LL1STB</i>	PLL1稳定中断标志 (CL型产品)
<i>RCU_INT_FLAG_P</i> <i>LL2STB</i>	PLL2稳定中断标志 (CL型产品)
<i>RCU_INT_FLAG_C</i> <i>KM</i>	HXTAL时钟阻塞中断标志
<i>RCU_INT_FLAG_IR</i> <i>C48MSTB</i>	IRC48M稳定中断标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the clock stabilization interrupt flag */

if(SET == rcu_interrupt_flag_get(RCU_INT_FLAG_HXTALSTB)){
}
```

函数 **rcu_interrupt_flag_clear**

函数rcu_interrupt_flag_clear描述见下表：

表 3-509. 函数 **rcu_interrupt_flag_clear**

函数名称	rcu_interrupt_flag_clear
函数原形	void rcu_interrupt_flag_clear(rcu_int_flag_clear_enum int_flag);
功能描述	清除中断标志和时钟阻塞中断标志
先决条件	-
被调用函数	-

输入参数{in}	
int_flag	时钟稳定和阻塞中断标志清除，参考rcu_int_flag_clear_enum
<i>RCU_INT_FLAG_IR_C40KSTB_CLR</i>	IRC40K稳定中断标志清除
<i>RCU_INT_FLAG_L_XTALSTB_CLR</i>	LXTAL稳定中断标志清除
<i>RCU_INT_FLAG_IR_C8MSTB_CLR</i>	IRC8M稳定中断标志清除
<i>RCU_INT_FLAG_H_XTALSTB_CLR</i>	HXTAL稳定中断标志清除
<i>RCU_INT_FLAG_P_LLSTB_CLR</i>	PLL稳定中断标志清除
<i>RCU_INT_FLAG_P_LL1STB_CLR</i>	PLL1稳定中断标志清除（CL型产品）
<i>RCU_INT_FLAG_P_LL2STB_CLR</i>	PLL2稳定中断标志清除（CL型产品）
<i>RCU_INT_FLAG_C_KM_CLR</i>	时钟阻塞中断标志清除
<i>RCU_INT_FLAG_IR_C48MSTB_CLR</i>	IRC48M稳定中断标志清除
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt HXTAL stabilization interrupt flag */
rcu_interrupt_flag_clear(RCU_INT_FLAG_HXTALSTB_CLR);
```

函数 **rcu_interrupt_enable**

函数**rcu_interrupt_enable**描述见下表：

表 3-510. 函数 **rcu_interrupt_enable**

函数名称	rcu_interrupt_enable

函数原形	void rcu_interrupt_enable(rcu_int_enum interrupt);
功能描述	使能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断
RCU_INT_IRC40KS_TB	IRC40K稳定中断
RCU_INT_LXTALS_TB	LXTAL稳定中断
RCU_INT_IRC8MS_TB	IRC8M稳定中断使能
RCU_INT_HXTALS_TB	HXTAL稳定中断
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断 (CL型产品)
RCU_INT_PLL2STB	PLL2稳定中断 (CL型产品)
RCU_INT_IRC48M_STB	IRC48M稳定中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL stabilization interrupt */
rcu_interrupt_enable(RCU_INT_HXTALSTB);
```

函数 **rcu_interrupt_disable**

函数rcu_interrupt_disable描述见下表：

表 3-511. 函数 **rcu_interrupt_disable**

函数名称	rcu_interrupt_disable
------	-----------------------

函数原形	void rcu_interrupt_disable(rcu_int_enum interrupt);
功能描述	禁能时钟稳定中断
先决条件	-
被调用函数	-
输入参数{in}	
interrupt	时钟稳定中断
RCU_INT_IRC40KS_TB	IRC40K稳定中断
RCU_INT_LXTALS_TB	LXTAL稳定中断
RCU_INT_IRC8MS_TB	IRC8M稳定中断使能
RCU_INT_HXTALS_TB	HXTAL稳定中断
RCU_INT_PLLSTB	PLL稳定中断
RCU_INT_PLL1STB	PLL1稳定中断 (CL型产品)
RCU_INT_PLL2STB	PLL2稳定中断 (CL型产品)
RCU_INT_IRC48M_STB	IRC48M稳定中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL stabilization interrupt */
rcu_interrupt_disable(RCU_INT_HXTALSTB);
```

函数 **rcu_lxtal_drive_capability_config**

函数rcu_lxtal_drive_capability_config描述见下表：

表 3-512. 函数 **rcu_lxtal_drive_capability_config**

函数名称	rcu_lxtal_drive_capability_config
------	-----------------------------------

函数原形	void rcu_lxtal_drive_capability_config(uint32_t lxtal_dricap);
功能描述	配置LXTAL驱动能力
先决条件	-
被调用函数	-
输入参数{in}	
lxtal_dricap	LXTAL驱动能力
<i>RCU_LXTAL_LOW_DRI</i>	低驱动能力
<i>RCU_LXTAL_MED_LOWDRI</i>	中低驱动能力
<i>RCU_LXTAL_MED_HIGHDRI</i>	中高驱动能力
<i>RCU_LXTAL_HIGH_DRI</i>	高驱动能力
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the LXTAL drive capability */
rcu_lxtal_drive_capability_config (RCU_LXTAL_LOWDRI);
```

函数 **rcu_osc_stab_wait**

函数rcu_osc_stab_wait描述见下表：

表 3-513. 函数 **rcu_osc_stab_wait**

函数名称	rcu_osc_stab_wait
函数原形	ErrStatus rcu_osc_stab_wait(rcu_osc_type_enum osci);
功能描述	等待振荡器稳定标志位置位或振荡器起振超时
先决条件	-
被调用函数	rcu_flag_get

输入参数{in}	
osci	振荡器类型, 参考rcu_osc_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器
<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
<i>RCU_PLL1_CK</i>	锁相环1 (CL型产品)
<i>RCU_PLL2_CK</i>	锁相环2 (CL型产品)
输出参数{out}	
-	-
返回值	
ErrStatus	SUCCESS 或 ERROR

例如:

```
/* wait for oscillator stabilization flag */

if(SUCCESS == rcu_osc_stab_wait(RCU_HXTAL)){
}
```

函数 **rcu_osc_on**

函数rcu_osc_on描述见下表:

表 3-514. 函数 **rcu_osc_on**

函数名称	rcu_osc_on
函数原形	void rcu_osc_on(rcu_osc_type_enum osci);
功能描述	打开振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型, 参考rcu_osc_type_enum

<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器
<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
<i>RCU_PLL1_CK</i>	锁相环1 (CL型产品)
<i>RCU_PLL2_CK</i>	锁相环2 (CL型产品)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn on the high speed crystal oscillator */
rcu_osci_on(RCU_HXTAL);
```

函数 **rcu_osci_off**

函数rcu_osci_off描述见下表：

表 3-515. 函数 **rcu_osci_off**

函数名称	rcu_osci_off
函数原形	void rcu_osci_off(rcu_osci_type_enum osci);
功能描述	关闭振荡器
先决条件	-
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考rcu_osci_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
<i>RCU_IRC8M</i>	内部8M RC振荡器

<i>RCU_IRC48M</i>	内部48M RC振荡器
<i>RCU_IRC40K</i>	内部40K RC振荡器
<i>RCU_PLL_CK</i>	锁相环
<i>RCU_PLL1_CK</i>	锁相环1 (CL型产品)
<i>RCU_PLL2_CK</i>	锁相环2 (CL型产品)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off the high speed crystal oscillator */
rcu_osc_i_off(RCU_HXTAL);
```

函数 **rcu_osc_i_bypass_mode_enable**

函数rcu_osc_i_bypass_mode_enable描述见下表：

表 3-516. 函数 **rcu_osc_i_bypass_mode_enable**

函数名称	rcu_osc_i_bypass_mode_enable
函数原形	void rcu_osc_i_bypass_mode_enable(rcu_osc_i_type_enum osci);
功能描述	使能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考rcu_osc_i_type_enum
<i>RCU_HXTAL</i>	高速晶体振荡器
<i>RCU_LXTAL</i>	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the high speed crystal oscillator bypass mode */
```

```
rcu_osc_bypass_mode_enable(RCU_HXTAL);
```

函数 **rcu_osc_bypass_mode_disable**

函数rcu_osc_bypass_mode_disable描述见下表：

表 3-517. 函数 **rcu_osc_bypass_mode_disable**

函数名称	rcu_osc_bypass_mode_disable
函数原形	void rcu_osc_bypass_mode_disable(rcu_osc_type_enum osci);
功能描述	禁能振荡器时钟旁路模式
先决条件	HXTALEN或LXTALEN应在使能振荡器时钟旁路模式前先复位
被调用函数	-
输入参数{in}	
osci	振荡器类型，参考rcu_osc_type_enum
RCU_HXTAL	高速晶体振荡器
RCU_LXTAL	低速晶体振荡器
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the high speed crystal oscillator bypass mode */
```

```
rcu_osc_bypass_mode_disable(RCU_HXTAL);
```

函数 **rcu_hxtal_clock_monitor_enable**

函数rcu_hxtal_clock_monitor_enable描述见下表：

表 3-518. 函数 **rcu_hxtal_clock_monitor_enable**

函数名称	rcu_hxtal_clock_monitor_enable
函数原形	void rcu_hxtal_clock_monitor_enable(void);
功能描述	使能HXTAL时钟监视器

先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_enable();
```

函数 **rcu_hxtal_clock_monitor_disable**

函数rcu_hxtal_clock_monitor_disable描述见下表：

表 3-519. 函数 **rcu_hxtal_clock_monitor_disable**

函数名称	rcu_hxtal_clock_monitor_disable
函数原形	void rcu_hxtal_clock_monitor_disable(void);
功能描述	禁能HXTAL时钟监视器
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the HXTAL clock monitor */

rcu_hxtal_clock_monitor_disable();
```

函数 rcu_irc8m_adjust_value_set

函数rcu_irc8m_adjust_value_set描述见下表:

表 3-520. 函数 rcu_irc8m_adjust_value_set

函数名称	rcu_irc8m_adjust_value_set
函数原形	void rcu_irc8m_adjust_value_set(uint32_t irc8m_adjval);
功能描述	设置内部8MHz RC振荡器时钟调整值
先决条件	-
被调用函数	-
输入参数{in}	
irc8m_adjval	IRC8M调整值（0到0x1F之间）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set the IRC8M adjust value */
rcu_irc8m_adjust_value_set(0x10);
```

函数 rcu_deepsleep_voltage_set

函数rcu_deepsleep_voltage_set描述见下表:

表 3-521. 函数 rcu_deepsleep_voltage_set

函数名称	rcu_deepsleep_voltage_set
函数原形	void rcu_deepsleep_voltage_set(uint32_t dsvol);
功能描述	设置深度睡眠模式电压值
先决条件	-
被调用函数	-
输入参数{in}	
dsvol	深度睡眠模式电压值
RCU_DEEPSLEEP	在深度睡眠模式下内核电压为1.0V

<i>_V_1_0</i>	
<i>RCU_DEEPSLEEP_V_0_9</i>	在深度睡眠模式下内核电压为0.9V
<i>RCU_DEEPSLEEP_V_0_8</i>	在深度睡眠模式下内核电压为0.8V
<i>RCU_DEEPSLEEP_V_0_7</i>	在深度睡眠模式下内核电压为0.7V
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the deep-sleep mode voltage */

rcu_deepsleep_voltage_set(RCU_DEEPSLEEP_V_1_0);
```

函数 **rcu_clock_freq_get**

函数**rcu_clock_freq_get**描述见下表：

表 3-522. 函数 **rcu_clock_freq_get**

函数名称	rcu_clock_freq_get
函数原形	uint32_t rcu_clock_freq_get(rcu_clock_freq_enum clock);
功能描述	获取系统时钟、总线频率
先决条件	-
被调用函数	-
输入参数{in}	
clock	要获取的时钟频率
CK_SYS	系统时钟
CK_AHB	AHB时钟
CK_APB1	APB1时钟
CK_APB2	APB2时钟
输出参数{out}	

-	-
返回值	
ck_freq	系统时钟/AHB时钟/APB1时钟/APB2时钟频率

例如：

```
uint32_t temp_freq;  
  
/* get the system clock frequency */  
  
temp_freq = rcu_clock_freq_get(CK_SYS);
```

3.20. RTC

实时时钟RTC通常被用作时钟日历。位于备份域中的RTC电路，包含一个32位的累加计数器、一个闹钟、一个预分频器、一个分频器以及RTC时钟配置寄存器。章节[3.20.1](#)描述了RTC的寄存器列表，章节[3.20.2](#)对RTC库函数进行说明。

3.20.1. 外设寄存器描述

RTC寄存器列表如下表所示：

表 3-523. RTC 寄存器

寄存器名称	寄存器描述
RTC_INTEN	中断使能寄存器
RTC_CTL	控制寄存器
RTC_PSCH	预分频寄存器高位
RTC_PSCL	预分频寄存器低位
RTC_DIVH	分频寄存器高位
RTC_DIVL	分频寄存器低位
RTC_CNTH	计数寄存器高位
RTC_CNTL	计数寄存器低位
RTC_ALRMH	闹钟寄存器高位
RTC_ALRML	闹钟寄存器低位

3.20.2. 外设库函数描述

RTC库函数列表如下表所示：

表 3-524.RTC 库函数

库函数名称	库函数描述
rtc_interrupt_enable	使能RTC中断
rtc_interrupt_disable	失能RTC中断
rtc_configuration_mode_enter	进入RTC配置模式
rtc_configuration_mode_exit	退出RTC配置模式
rtc_lwoff_wait	等待最近一次对RTC寄存器的写操作完成
rtc_register_sync_wait	等待RTC寄存器(RTC_CNTx、RTC_ALRMx 和 RTC_PSCx)与 RTC 的 APB 时钟同步
rtc_counter_get	获取RTC计数器的值
rtc_counter_set	设置RTC计数器的值
rtc_prescaler_set	设置RTC预分频值
rtc_alarm_config	设置RTC闹钟值
rtc_divider_get	获取RTC分频值
rtc_flag_get	获取RTC标志位状态
rtc_flag_clear	清除RTC标志位状态

函数 rtc_interrupt_enable

函数 rtc_interrupt_enable 描述见下表:

表 3-525. 函数 rtc_interrupt_enable

函数名称	rtc_interrupt_enable
函数原型	void rtc_interrupt_enable(uint32_t interrupt);
功能描述	使能RTC中断
先决条件	调用此函数之前，必须调用函数 rtc_lwoff_wait() (等待标志位LWOFF置位)
被调用函数	-
输入参数{in}	
interrupt	待使能的RTC中断源
RTC_INT_SECOND	秒中断
RTC_INT_ALARM	闹钟中断
RTC_INT_OVERFL	溢出中断

OW	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* enable the RTC second interrupt */

rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 **rtc_interrupt_disable**

函数 **rtc_interrupt_disable** 描述见下表：

表 3-526. 函数 **rtc_interrupt_disable**

函数名称	rtc_interrupt_disable
函数原型	void rtc_interrupt_disable(uint32_t interrupt);
功能描述	失能RTC中断
先决条件	调用此函数之前，必须调用函数 <code>rtc_lwoff_wait()</code> （等待标志位LWOFF置位）
被调用函数	-
输入参数{in}	
interrupt	待失能的RTC中断源
RTC_INT_SECOND	秒中断
RTC_INT_ALARM	闹钟中断
RTC_INT_OVERFLOW	溢出中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait( );

/* disable the RTC second interrupt */

rtc_interrupt_disable(RTC_INT_SECOND);

```

函数 **rtc_configuration_mode_enter**

函数`rtc_configuration_mode_enter`描述见下表：

表 3-527. 函数 `rtc_configuration_mode_enter`

函数名称	rtc_configuration_mode_enter
函数原型	<code>void rtc_configuration_mode_enter(void);</code>
功能描述	进入RTC配置模式
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* enter RTC configuration mode */

rtc_configuration_mode_enter( );

```

函数 **rtc_configuration_mode_exit**

函数`rtc_configuration_mode_exit`描述见下表：

表 3-528. 函数 `rtc_configuration_mode_exit`

函数名称	rtc_configuration_mode_exit
函数原型	<code>void rtc_configuration_mode_exit(void);</code>
功能描述	退出RTC配置模式
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* exit RTC configuration mode */
rtc_configuration_mode_exit();
```

函数 **rtc_lwoff_wait**

函数**rtc_lwoff_wait**描述见下表：

表 3-529. 函数 **rtc_lwoff_wait**

函数名称	rtc_lwoff_wait
函数原型	void rtc_lwoff_wait(void);
功能描述	等待最近一次对RTC寄存器的写操作完成
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */
rtc_lwoff_wait();
/* enable the RTC second interrupt */
```

```
rtc_interrupt_enable(RTC_INT_SECOND);
```

函数 **rtc_register_sync_wait**

函数`rtc_register_sync_wait`描述见下表：

表 3-530. 函数 `rtc_register_sync_wait`

函数名称	rtc_register_sync_wait
函数原型	<code>void rtc_register_sync_wait(void);</code>
功能描述	等待RTC寄存器(RTC_CNTx、RTC_ALRMx和RTC_PSCx)与RTC的APB时钟同步
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait for RTC registers synchronization */

rtc_register_sync_wait();
```

函数 **rtc_counter_get**

函数`rtc_counter_get`描述见下表：

表 3-531. 函数 `rtc_counter_get`

函数名称	rtc_counter_get
函数原型	<code>uint32_t rtc_counter_get(void);</code>
功能描述	获取RTC计时器的值
先决条件	-
被调用函数	-
输入参数{in}	

-	-
输出参数{out}	
RTC counter value	RTC计时器的值
返回值	
-	-

例如：

```
/* get the counter value */

uint32_t rtc_counter_value;

rtc_counter_value = rtc_counter_get();
```

函数 **rtc_counter_set**

函数 **rtc_counter_set** 描述见下表：

表 3-532. Function **rtc_counter_set**

函数名称	rtc_counter_set
函数原型	void rtc_counter_set(uint32_t cnt);
功能描述	设置RTC计数器的值
先决条件	-
被调用函数	-
输入参数{in}	
cnt	RTC计数器的值 (0-0xFFFF FFFF)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set counter value to 0xFFFF */

rtc_counter_set (0xFFFF);
```

函数 rtc_prescaler_set

函数rtc_prescaler_set描述见下表:

表 3-533. 函数 rtc_prescaler_set

函数名称	rtc_interrupt_rtc_prescaler_set
函数原型	void rtc_prescaler_set(uint32_t psc);
功能描述	设置RTC预分频值
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait()（等待标志位LWOFF置位）
被调用函数	rtc_configuration_mode_enter / rtc_configuration_mode_exit
输入参数{in}	
psc	RTC预分频值（0-0x000F FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set RTC prescaler value to 0x7FFF */

rtc_prescaler_set(0x7FFF);
```

函数 rtc_alarm_config

函数rtc_alarm_config描述见下表:

表 3-534. 函数 rtc_alarm_config

函数名称	rtc_alarm_config
函数原型	void rtc_alarm_config(uint32_t alarm);
功能描述	设置RTC闹钟值
先决条件	调用此函数之前，必须调用函数rtc_lwoff_wait()（等待标志位LWOFF置位）
被调用函数	rtc_configuration_mode_enter / rtc_configuration_mode_exit
输入参数{in}	

alarm	RTC闹钟值（0-0xFFFF FFFF）
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* wait until last write operation on RTC registers has finished */

rtc_lwoff_wait();

/* set alarm value to 0xFFFF */

rtc_alarm_config (0xFFFF);
```

函数 **rtc_divider_get**

函数`rtc_divider_get`描述见下表：

表 3-535. 函数 **rtc_divider_get**

函数名称	rtc_divider_get
函数原型	uint32_t rtc_divider_get(void);
功能描述	获取RTC分频值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
RTC divider value	RTC分频值

例如：

```
/* get the current RTC divider value */

uint32_t rtc_divider_value;

rtc_divider_value = rtc_divider_get();
```

函数 `rtc_flag_get`

函数`rtc_flag_get`描述见下表:

表 3-536. 函数 `rtc_flag_get`

函数名称	<code>rtc_flag_get</code>
函数原型	<code>FlagStatus rtc_flag_get(uint32_t flag);</code>
功能描述	获取RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
<code>flag</code>	需要获取的RTC标志位
<code>RTC_FLAG_SECOND</code>	秒中断标志位
<code>RTC_FLAG_ALARM</code>	闹钟中断标志位
<code>RTC_FLAG_OVERFLOW</code>	溢出中断标志位
<code>RTC_FLAG_RSYN</code>	寄存器同步标志位
<code>RTC_FLAG_LWOF</code>	最近一次对RTC寄存器的写操作完成标志位
输出参数{out}	
-	-
返回值	
<code>FlagStatus</code>	SET 或 RESET

例如:

```
/* get the RTC overflow interrupt status */

FlagStatus alarm_status;

alarm_status = rtc_flag_get (RTC_FLAG_ALARM);
```

函数 `rtc_flag_clear`

函数`rtc_flag_clear`描述见下表:

表 3-537. 函数 `rtc_flag_clear`

函数名称	<code>rtc_flag_clear</code>
------	-----------------------------

函数原型	void rtc_flag_clear(uint32_t flag);
功能描述	清除RTC标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	待清除的RTC标志位
RTC_FLAG_SECOND	秒中断标志位
RTC_FLAG_ALARM	闹钟中断标志位
RTC_FLAG_OVERRLOW	溢出中断标志位
RTC_FLAG_RSYN	寄存器同步标志位
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the RTC alarm flag */

rtc_flag_clear (RTC_FLAG_ALARM);
```

3.21. SDIO

安全的数字输入/输出接口（SDIO）定义了SD卡、SD I/O卡、多媒体卡（MMC）和CE-ATA卡主机接口，提供AHB系统总线与SD存储卡、SD I/O卡、MMC和CE-ATA设备之间的数据传输。章节[3.21.1](#)描述了SDIO的寄存器列表，章节[3.21.2](#)对SDIO库函数进行说明。

3.21.1. 外设寄存器说明

SDIO寄存器列表如下表所示：

表 3-538. SDIO 寄存器

寄存器名称	寄存器描述
SDIO_PWRCTL	电源控制寄存器

寄存器名称	寄存器描述
SDIO_CLKCTL	时钟控制寄存器
SDIO_CMDAGMT	命令参数寄存器
SDIO_CMDCTL	命令控制寄存器
SDIO_RSPCMDIDX	命令索引响应寄存器
SDIO_RESPx x=0..3	响应寄存器
SDIO_DATATO	数据超时寄存器
SDIO_DATALEN	数据长度寄存器
SDIO_DATACTL	数据控制寄存器
SDIO_DATACNT	数据计数寄存器
SDIO_STAT	状态寄存器
SDIO_INTC	中断清除寄存器
SDIO_INTEN	中断使能寄存器
SDIO_FIFOCNT	FIFO计数寄存器
SDIO_FIFO	FIFO数据寄存器

3.21.2. 外设库函数说明

SDIO库函数列表如下表所示：

表 3-539. SDIO 库函数

库函数名称	库函数描述
sdio_deinit	复位SDIO
sdio_clock_config	配置SDIO时钟
sdio_hardware_clock_enable	使能硬件时钟控制
sdio_hardware_clock_disable	禁能硬件时钟控制
sdio_bus_mode_set	设置多种SDIO卡总线模式
sdio_power_state_set	设置SDIO电源状态
sdio_power_state_get	获取SDIO电源状态
sdio_clock_enable	使能SDIO_CLK时钟

库函数名称	库函数描述
sdio_clock_disable	禁能SDIO_CLK时钟
sdio_command_response_config	配置命令和响应
sdio_wait_type_set	设置命令状态机等待类型
sdio_csm_enable	使能命令状态机
sdio_csm_disable	禁能命令状态机
sdio_command_index_get	获取上一次响应的命令索引
sdio_response_get	获取上一次响应的接收命令
sdio_data_config	配置数据超时、数据长度和数据块大小
sdio_data_transfer_config	配置数据传输模式和方向
sdio_dsm_enable	使能数据传输的数据状态机
sdio_dsm_disable	禁能数据传输的数据状态机
sdio_data_write	在发送FIFO里写入数据（一个字）
sdio_data_read	在接收FIFO里读取数据（一个字）
sdio_data_counter_get	获取要传输到卡的剩余数据字节的数目
sdio_fifo_counter_get	从FIFO中获取要写入或读取的字数
sdio_dma_enable	使能SDIO的DMA请求
sdio_dma_disable	禁能SDIO的DMA请求
sdio_flag_get	获取SDIO的标志位状态
sdio_flag_clear	清除SDIO的标志位状态
sdio_interrupt_enable	使能SDIO中断
sdio_interrupt_disable	禁能SDIO中断
sdio_interrupt_flag_get	获取SDIO的中断标志位状态
sdio_interrupt_flag_clear	清除SDIO的中断标志位状态
sdio_readwait_enable	使能读等待模式（仅限SD I/O模式）
sdio_readwait_disable	禁能读等待模式（仅限SD I/O模式）
sdio_stop_readwait_enable	使能停止读等待过程的功能（仅限SD I/O模式）
sdio_stop_readwait_disable	禁能停止读等待过程的功能（仅限SD I/O模式）

库函数名称	库函数描述
sdio_readwait_type_set	设置读等待类型(仅限SD I/O模式)
sdio_operation_enable	使能SD I/O模式特定操作(仅限SD I/O模式)
sdio_operation_disable	禁能SD I/O模式特定操作(仅限SD I/O模式)
sdio_suspend_enable	使能SD I/O暂停模式(仅限SD I/O模式)
sdio_suspend_disable	禁能SD I/O暂停模式(仅限SD I/O模式)
sdio_ceata_command_enable	使能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_command_disable	禁能CE-ATA命令(仅限CE-ATA模式)
sdio_ceata_interrupt_enable	使能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_interrupt_disable	禁能CE-ATA中断(仅限CE-ATA模式)
sdio_ceata_command_completion_enable	使能CE-ATA命令完成信号(仅限CE-ATA模式)
sdio_ceata_command_completion_disable	禁能CE-ATA命令完成信号(仅限CE-ATA模式)

函数 **sdio_deinit**

函数sdio_deinit描述见下表:

表 3-540. 函数 **sdio_deinit**

函数名称	sdio_deinit
函数原形	void sdio_deinit(void);
功能描述	复位SDIO
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* deinitialize the SDIO */
```

```
sdio_deinit();
```

函数 **sdio_clock_config**

函数**sdio_clock_config**描述见下表：

表 3-541. 函数 **sdio_clock_config**

函数名称	sdio_clock_config
函数原形	<code>void sdio_clock_config(uint32_t clock_edge, uint32_t clock_bypass, uint32_t clock_powersave, uint16_t clock_division);</code>
功能描述	配置SDIO时钟
先决条件	-
被调用函数	-
输入参数{in}	
clock_edge	SDIO_CLK时钟边沿选择
SDIO_SDIOCLKED GE_RISING	选择SDIOCLK的上升沿产生SDIO_CLK
SDIO_SDIOCLKED GE_FALLING	选择SDIOCLK的下降沿产生SDIO_CLK
输入参数{in}	
clock_bypass	旁路时钟使能
SDIO_CLOCKBYPASS SS_ENABLE	使能旁路时钟
SDIO_CLOCKBYPASS SS_DISABLE	失能旁路时钟
输入参数{in}	
clock_powersave	SDIO_CLK时钟动态开启/关闭以节省功耗
SDIO_CLOCKPWR SAVE_ENABLE	SDIO_CLK时钟在总线空闲时关闭
SDIO_CLOCKPWR SAVE_DISABLE	SDIO_CLK时钟总是开启
输入参数{in}	
clock_division	时钟分频，小于512

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the SDIO clock */
sdio_clock_config(SDIO_SDIOCLKEDGE_RISING,      SDIO_CLOCKBYPASS_DISABLE,
SDIO_CLOCKPWRSAVE_DISABLE, SD_CLK_DIV_TRANS);
```

函数 **sdio_hardware_clock_enable**

函数sdio_hardware_clock_enable描述见下表：

表 3-542. 函数 **sdio_hardware_clock_enable**

函数名称	sdio_hardware_clock_enable
函数原形	void sdio_hardware_clock_enable(void);
功能描述	使能硬件时钟控制
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable hardware clock control */

sdio_hardware_clock_enable();
```

函数 **sdio_hardware_clock_disable**

函数sdio_hardware_clock_disable描述见下表：

表 3-543. 函数 `sdio_hardware_clock_disable`

函数名称	<code>sdio_hardware_clock_disable</code>
函数原形	<code>void sdio_hardware_clock_disable(void);</code>
功能描述	禁能硬件时钟控制
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable hardware clock control */

sdio_hardware_clock_disable();
```

函数 `sdio_bus_mode_set`

函数 `sdio_bus_mode_set` 描述见下表：

表 3-544. 函数 `sdio_bus_mode_set`

函数名称	<code>sdio_bus_mode_set</code>
函数原形	<code>void sdio_bus_mode_set(uint32_t bus_mode);</code>
功能描述	设置多种SDIO卡总线模式
先决条件	-
被调用函数	-
输入参数{in}	
bus_mode	SDIO卡总线模式
SDIO_BUSMODE_1 BIT	1位SDIO卡总线模式
SDIO_BUSMODE_4 BIT	4位SDIO卡总线模式

<i>SDIO_BUSMODE_8BIT</i>	8位SDIO卡总线模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO bus mode */
sdio_bus_mode_set(SDIO_BUSMODE_1BIT);
```

函数 **sdio_power_state_set**

函数sdio_power_state_set描述见下表：

表 3-545. 函数 **sdio_power_state_set**

函数名称	sdio_power_state_set
函数原形	void sdio_power_state_set(uint32_t power_state);
功能描述	设置SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
power_state	SDIO电源状态
<i>SDIO_POWER_ON</i>	SDIO上电
<i>SDIO_POWER_OF_F</i>	SDIO断电
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SDIO power state */
sdio_power_state_set(SDIO_POWER_ON);
```

函数 **sdio_power_state_get**

函数 **sdio_power_state_get** 描述见下表：

表 3-546. 函数 **sdio_power_state_get**

函数名称	sdio_power_state_get
函数原形	uint32_t sdio_power_state_get(void);
功能描述	获取SDIO电源状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	SDIO_POWER_ON / SDIO_POWER_OFF

例如：

```
/* get the SDIO power state */

uint32_t sdio_power_value;

sdio_power_value = sdio_power_state_get();
```

函数 **sdio_clock_enable**

函数 **sdio_clock_enable** 描述见下表：

表 3-547. 函数 **sdio_clock_enable**

函数名称	sdio_clock_enable
函数原形	void sdio_clock_enable(void);
功能描述	使能SDIO_CLK时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SDIO_CLK clock output */
sdio_clock_enable();
```

函数 **sdio_clock_disable**

函数sdio_clock_disable描述见下表：

表 3-548. 函数 **sdio_clock_disable**

函数名称	sdio_clock_disable
函数原形	void sdio_clock_disable(void);
功能描述	禁能SDIO_CLK时钟
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SDIO_CLK clock output */
sdio_clock_disable();
```

函数 **sdio_command_response_config**

函数sdio_command_response_config描述见下表：

表 3-549. 函数 **sdio_command_response_config**

函数名称	sdio_command_response_config
------	------------------------------

函数原形	void sdio_command_response_config(uint32_t cmd_index, uint32_t cmd_argument, uint32_t response_type);
功能描述	配置命令和响应
先决条件	-
被调用函数	-
输入参数{in}	
cmd_index	命令索引，请参阅相关规范
输入参数{in}	
cmd_argument	命令参数，请参阅相关规范
输入参数{in}	
response_type	命令响应类型
SDIO_RESPONSETYPE_NO	无响应
SDIO_RESPONSETYPE_SHORT	短响应
SDIO_RESPONSETYPE_LONG	长响应
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* CMD2(SD_CMD_ALL_SEND_CID) command response config*/
sdio_command_response_config(SD_CMD_ALL_SEND_CID, (uint32_t)0x0,
  SDIO_RESPONSETYPE_LONG);
```

函数 **sdio_wait_type_set**

函数sdio_wait_type_set描述见下表：

表 3-550. 函数 **sdio_wait_type_set**

函数名称	sdio_wait_type_set
函数原形	void sdio_wait_type_set(uint32_t wait_type);

功能描述	设置命令状态机等待类型
先决条件	-
被调用函数	-
输入参数{in}	
wait_type	等待类型
<i>SDIO_WAITTYPE_NO</i>	不等待中断
<i>SDIO_WAITTYPE_INTERRUPT</i>	等待中断
<i>SDIO_WAITTYPE_DATAEND</i>	等待数据传输结束
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the command state machine wait type */
sdio_wait_type_set(SDIO_WAITTYPE_NO);
```

函数 **sdio_csm_enable**

函数sdio_csm_enable描述见下表：

表 3-551. 函数 **sdio_csm_enable**

函数名称	sdio_csm_enable
函数原形	void sdio_csm_enable(void);
功能描述	使能命令状态机
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the CSM(command state machine) */

sdio_csm_enable();
```

函数 **sdio_csm_disable**

函数sdio_csm_disable描述见下表：

表 3-552. 函数 **sdio_csm_disable**

函数名称	sdio_csm_disable
函数原形	void sdio_csm_disable(void);
功能描述	禁能命令状态机
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CSM(command state machine) */

sdio_csm_disable();
```

函数 **sdio_command_index_get**

函数sdio_command_index_get描述见下表：

表 3-553. 函数 **sdio_command_index_get**

函数名称	sdio_command_index_get
函数原形	uint8_t sdio_command_index_get(void);

功能描述	获取上一次响应的命令索引
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint8_t	上一次响应的命令索引

例如：

```
/* get SDIO command index */
uint8_t sdio_command_value;
sdio_command_value = sdio_command_index_get();
```

函数 **sdio_response_get**

函数sdio_response_get描述见下表：

表 3-554. 函数 **sdio_response_get**

函数名称	sdio_response_get
函数原形	uint32_t sdio_response_get(uint32_t response);
功能描述	获取上一次响应的接收命令
先决条件	-
被调用函数	-
输入参数{in}	
response	SDIO响应
SDIO_RESPONSE0	卡响应 [31:0]/卡响应 [127:96]
SDIO_RESPONSE1	卡响应 [95:64]
SDIO_RESPONSE2	卡响应 [63:32]
SDIO_RESPONSE3	卡响应 [31:1], 加上位0
输出参数{out}	

-	-
返回值	
uint32_t	上一次响应的接收命令

例如：

```
/* store the CID0 numbers */

uint32_t sdio_cid[0];

sdio_cid[0] = sdio_response_get(SDIO_RESPONSE0);
```

函数 **sdio_data_config**

函数sdio_data_config描述见下表：

表 3-555. 函数 **sdio_data_config**

函数名称	sdio_data_config
函数原形	void sdio_data_config(uint32_t data_timeout, uint32_t data_length, uint32_t data_blocksize);
功能描述	配置数据超时、数据长度和数据块大小
先决条件	-
被调用函数	-
输入参数{in}	
data_timeout	卡总线时钟周期中的数据超时周期
输入参数{in}	
data_length	要传输的数据字节数
输入参数{in}	
data_blocksize	块传输中数据块的大小
SDIO_DATABLOCK_SIZE_1BYTE	块大小 = 1字节
SDIO_DATABLOCK_SIZE_2BYTES	块大小 = 2字节
SDIO_DATABLOCK_SIZE_4BYTES	块大小 = 4字节
SDIO_DATABLOCK_SIZE_8BYTES	块大小 = 8字节

<code>SDIO_DATABLOCK_SIZE_16BYTES</code>	块大小 = 16字节
<code>SDIO_DATABLOCK_SIZE_32BYTES</code>	块大小 = 32字节
<code>SDIO_DATABLOCK_SIZE_64BYTES</code>	块大小 = 64字节
<code>SDIO_DATABLOCK_SIZE_128BYTES</code>	块大小 = 128字节
<code>SDIO_DATABLOCK_SIZE_256BYTES</code>	块大小 = 256字节
<code>SDIO_DATABLOCK_SIZE_512BYTES</code>	块大小 = 512字节
<code>SDIO_DATABLOCK_SIZE_1024BYTES</code>	块大小 = 1024字节
<code>SDIO_DATABLOCK_SIZE_2048BYTES</code>	块大小 = 2048字节
<code>SDIO_DATABLOCK_SIZE_4096BYTES</code>	块大小 = 4096字节
<code>SDIO_DATABLOCK_SIZE_8192BYTES</code>	块大小 = 8192字节
<code>SDIO_DATABLOCK_SIZE_16384BYTES</code>	块大小 = 16384字节
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SDIO data */
sdio_data_config(0, 0, SDIO_DATABLOCKSIZE_1BYTE);
```

函数 `sdio_data_transfer_config`

函数`sdio_data_transfer_config`描述见下表：

表 3-556. 函数 `sdio_data_transfer_config`

函数名称	<code>sdio_data_transfer_config</code>
函数原形	<code>void sdio_data_transfer_config(uint32_t transfer_mode, uint32_t transfer_direction);</code>
功能描述	配置数据传输模式和方向
先决条件	-
被调用函数	-
输入参数{in}	
<code>transfer_mode</code>	数据传输模式
<code>SDIO_TRANSMODE_BLOCK</code>	块传输模式
<code>SDIO_TRANSMODE_STREAM</code>	流传输或SDIO多字节传输模式
输入参数{in}	
<code>transfer_direction</code>	数据传输方向
<code>SDIO_TRANS_DIRECTION_TOCARD</code>	写数据到卡上
<code>SDIO_TRANS_DIRECTION_TOSDIO</code>	从卡中读取数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SDIO data transmission */

sdio_data_transfer_config(SDIO_TRANS_DIRECTION_TOSDIO,
                           SDIO_TRANSMODE_BLOCK);
```

函数 `sdio_dsm_enable`

函数`sdio_dsm_enable`描述见下表：

表 3-557. 函数 **sdio_dsm_enable**

函数名称	sdio_dsm_enable
函数原形	void sdio_dsm_enable(void);
功能描述	使能数据传输的数据状态机
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the DSM(data state machine) */
sdio_dsm_enable();
```

函数 **sdio_dsm_disable**

函数sdio_dsm_disable描述见下表：

表 3-558. 函数 **sdio_dsm_disable**

函数名称	sdio_dsm_disable
函数原形	void sdio_dsm_disable(void);
功能描述	禁能数据传输的数据状态机
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* disable the DSM(data state machine) */
sdio_dsm_disable();
```

函数 **sdio_data_write**

函数sdio_data_write描述见下表：

表 3-559. 函数 **sdio_data_write**

函数名称	sdio_data_write
函数原形	void sdio_data_write(uint32_t data);
功能描述	在发送FIFO里写入数据（一个字）
先决条件	-
被调用函数	-
输入参数{in}	
data	往卡里写入32位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* write data(one word) to the transmit FIFO */
sdio_data_write(0x0000 0001);
```

函数 **sdio_data_read**

函数sdio_data_read描述见下表：

表 3-560. 函数 **sdio_data_read**

函数名称	sdio_data_read
函数原形	uint32_t sdio_data_read(void);
功能描述	在接收FIFO里读取数据（一个字）
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	接收的数据

例如：

```
/* read data(one word) from the receive FIFO */
sdio_data_read();
```

函数 **sdio_data_counter_get**

函数sdio_data_counter_get描述见下表：

表 3-561. 函数 **sdio_data_counter_get**

函数名称	sdio_data_counter_get
函数原形	uint32_t sdio_data_counter_get(void);
功能描述	获取要传输到卡的剩余数据字节的数目
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	要传输的剩余数据字节数

例如：

```
/* get the number of remaining data bytes to be transferred to card */
uint32_t sdio_data_value;
sdio_data_value = sdio_data_counter_get();
```

函数 **sdio_fifo_counter_get**

函数sdio_fifo_counter_get描述见下表:

表 3-562. 函数 **sdio_fifo_counter_get**

函数名称	sdio_fifo_counter_get
函数原形	uint32_t sdio_fifo_counter_get(void);
功能描述	从FIFO中获取要写入或读取的字数
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
uint32_t	剩余字数

例如:

```
/* get the number of words remaining to be written or read from FIFO */
uint32_t sdio_fifo_value;
sdio_fifo_value = sdio_fifo_counter_get();
```

函数 **sdio_dma_enable**

函数sdio_dma_enable描述见下表:

表 3-563. 函数 **sdio_dma_enable**

函数名称	sdio_dma_enable
函数原形	void sdio_dma_enable(void);
功能描述	使能SDIO的DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO DMA */

sdio_dma_enable();
```

函数 **sdio_dma_disable**

函数sdio_dma_disable描述见下表：

表 3-564. 函数 **sdio_dma_disable**

函数名称	sdio_dma_disable
函数原形	void sdio_dma_disable(void);
功能描述	禁能SDIO的DMA请求
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SDIO DMA */

sdio_dma_disable();
```

函数 **sdio_flag_get**

函数sdio_flag_get描述见下表：

表 3-565. 函数 **sdio_flag_get**

函数名称	sdio_flag_get
------	---------------

函数原形	FlagStatus sdio_flag_get(uint32_t flag);
功能描述	获取SDIO的标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC ERR	命令响应已接收 (CRC检测失败)
SDIO_FLAG_DTCR CERR	数据块已发送/已接收 (CRC检测失败)
SDIO_FLAG_CMDT MOUT	命令响应超时
SDIO_FLAG_DTTM OUT	数据超时
SDIO_FLAG_TXUR E	发送FIFO下溢错误发生
SDIO_FLAG_RXOR E	接收FIFO上溢错误发生
SDIO_FLAG_CMDR ECV	命令响应已接收 (CRC检测通过)
SDIO_FLAG_CMDS END	命令已发送 (不需响应)
SDIO_FLAG_DTEN D	数据结束 (数据计数器, SDIO_DATACNT为零)
SDIO_FLAG_STBIT E	总线上起始位错误
SDIO_FLAG_DTBL KEND	数据块已发送/已接收 (CRC检测通过)
SDIO_FLAG_CMDR UN	正在传输命令
SDIO_FLAG_TXRU N	正在传输数据
SDIO_FLAG_RXRU	正在接收数据

<i>N</i>	
<i>SDIO_FLAG_TFH</i>	发送FIFO半空：至少还有8个字可被写入到FIFO中
<i>SDIO_FLAG_RFH</i>	接收FIFO半满：FIFO中至少还有8个字可被读取
<i>SDIO_FLAG_TFF</i>	发送FIFO为满
<i>SDIO_FLAG_RFF</i>	接收FIFO为满
<i>SDIO_FLAG_TFE</i>	发送FIFO为空
<i>SDIO_FLAG_RFE</i>	接收FIFO为空
<i>SDIO_FLAG_TXDT VAL</i>	发送FIFO中的数据有效
<i>SDIO_FLAG_RXDT VAL</i>	接收FIFO中的数据有效
<i>SDIO_FLAG_SDIOI NT</i>	SD I/O中断已接收
<i>SDIO_FLAG_ATAE ND</i>	CE-ATA命令完成信号已接收（仅用于CMD61）
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the flags state of SDIO */

FlagStatus flag_value;

flag_value = sdio_flag_get(SDIO_FLAG_RFH);
```

函数 **sdio_flag_clear**

函数sdio_flag_clear描述见下表：

表 3-566. 函数 **sdio_flag_clear**

函数名称	sdio_flag_clear
函数原形	void sdio_flag_clear(uint32_t flag);
功能描述	清除SDIO的标志位状态
先决条件	-

被调用函数	-
输入参数{in}	
flag	SDIO标志位状态
SDIO_FLAG_CCRC ERR	命令响应已接收 (CRC检测失败)
SDIO_FLAG_DTCR CERR	数据块已发送/已接收 (CRC检测失败)
SDIO_FLAG_CMDT MOUT	命令响应超时
SDIO_FLAG_DTTM OUT	数据超时
SDIO_FLAG_TXUR E	发送FIFO下溢错误发生
SDIO_FLAG_RXOR E	接收FIFO上溢错误发生
SDIO_FLAG_CMDR ECV	命令响应已接收 (CRC检测通过)
SDIO_FLAG_CMDS END	命令已发送 (不需响应)
SDIO_FLAG_DTEN D	数据结束 (数据计数器, SDIO_DATACNT为零)
SDIO_FLAG_STBIT E	总线上起始位错误
SDIO_FLAG_DTBL KEND	数据块已发送/已接收 (CRC检测通过)
SDIO_FLAG_SDIOI NT	SD I/O中断已接收
SDIO_FLAG_ATAE ND	CE-ATA命令完成信号已接收 (仅用于CMD61)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the pending flags of SDIO */
sdio_flag_clear(SDIO_FLAG_DTCRCERR);
```

函数 **sdio_interrupt_enable**

函数**sdio_interrupt_enable**描述见下表：

表 3-567. 函数 **sdio_interrupt_enable**

函数名称	sdio_interrupt_enable
函数原形	void sdio_interrupt_enable(uint32_t int_flag);
功能描述	使能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
SDIO_INT_CCRCE_RR	命令响应CRC错误中断
SDIO_INT_DTCRC_ERR	数据CRC错误中断
SDIO_INT_CMDTM_OUT	命令响应超时中断
SDIO_INT_DTTMO_UT	数据超时中断
SDIO_INT_TXURE	发送FIFO下溢错误中断
SDIO_INT_RXORE	接收FIFO上溢错误中断
SDIO_INT_CMDRE_CV	命令响应已接收中断
SDIO_INT_CMDSE_ND	命令已发送中断
SDIO_INT_DTEND	数据结束中断
SDIO_INT_STBITE	起始位错误中断
SDIO_INT_DTBLKE_ND	数据块已发送/已接收中断

<i>SDIO_INT_CMDRU N</i>	正在传输命令中断
<i>SDIO_INT_TXRUN</i>	正在传输数据中断
<i>SDIO_INT_RXRUN</i>	正在接收数据中断
<i>SDIO_INT_TFH</i>	发送FIFO半满中断
<i>SDIO_INT_RFH</i>	接收FIFO半满中断
<i>SDIO_INT_TFF</i>	发送FIFO满中断
<i>SDIO_INT_RFF</i>	接收FIFO满中断
<i>SDIO_INT_TFE</i>	发送FIFO空中断
<i>SDIO_INT_RFE</i>	接收FIFO空中断
<i>SDIO_INT_TXDTVA L</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_RXDTV AL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOIN T</i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEN D</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SDIO corresponding interrupts */

sdio_interrupt_enable(SDIO_INT_CCRCCERR | SDIO_INT_DTTMOUT | SDIO_INT_RXORE
| SDIO_INT_DTEND | SDIO_INT_STBITE);
```

函数 **sdio_interrupt_disable**

函数sdio_interrupt_disable描述见下表：

表 3-568. 函数 **sdio_interrupt_disable**

函数名称	sdio_interrupt_disable
------	------------------------

函数原形	void sdio_interrupt_disable(uint32_t int_flag);
功能描述	禁能SDIO中断
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
<i>SDIO_INT_CCRCE RR</i>	命令响应CRC错误中断
<i>SDIO_INT_DTCRC ERR</i>	数据CRC错误中断
<i>SDIO_INT_CMDTM OUT</i>	命令响应超时中断
<i>SDIO_INT_DTTMO UT</i>	数据超时中断
<i>SDIO_INT_TXURE</i>	发送FIFO下溢错误中断
<i>SDIO_INT_RXORE</i>	接收FIFO上溢错误中断
<i>SDIO_INT_CMDRE CV</i>	命令响应已接收中断
<i>SDIO_INT_CMDSE ND</i>	命令已发送中断
<i>SDIO_INT_DTEND</i>	数据结束中断
<i>SDIO_INT_STBITE</i>	起始位错误中断
<i>SDIO_INT_DTBLKE ND</i>	数据块已发送/已接收中断
<i>SDIO_INT_CMDRU N</i>	正在传输命令中断
<i>SDIO_INT_TXRUN</i>	正在传输数据中断
<i>SDIO_INT_RXRUN</i>	正在接收数据中断
<i>SDIO_INT_TFH</i>	发送FIFO半满中断
<i>SDIO_INT_RFH</i>	接收FIFO半满中断
<i>SDIO_INT_TFF</i>	发送FIFO满中断

<i>SDIO_INT_RFF</i>	接收FIFO满中断
<i>SDIO_INT_TFE</i>	发送FIFO空中断
<i>SDIO_INT_RFE</i>	接收FIFO空中断
<i>SDIO_INT_TXDTVAL</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_RXDTVAL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_SDIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_ATAEN</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SDIO interrupt */
sdio_interrupt_disable(SDIO_INT_DTCRCERR);
```

函数 **sdio_interrupt_flag_get**

函数sdio_interrupt_flag_get描述见下表：

表 3-569. 函数 **sdio_interrupt_flag_get**

函数名称	sdio_interrupt_flag_get
函数原形	FlagStatus sdio_interrupt_flag_get(uint32_t int_flag);
功能描述	获取SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态
<i>SDIO_INT_FLAG_CRCERR</i>	命令响应CRC错误中断

<i>SDIO_INT_FLAG_D</i>	数据CRC错误中断
<i>TCRCERR</i>	
<i>SDIO_INT_FLAG_C</i>	命令响应超时中断
<i>MDTMOUT</i>	
<i>SDIO_INT_FLAG_D</i>	数据超时中断
<i>TTMOUT</i>	
<i>SDIO_INT_FLAG_T</i>	发送FIFO下溢错误中断
<i>XURE</i>	
<i>SDIO_INT_FLAG_R</i>	接收FIFO上溢错误中断
<i>XORE</i>	
<i>SDIO_INT_FLAG_C</i>	命令响应已接收中断
<i>MDRECV</i>	
<i>SDIO_INT_FLAG_C</i>	命令已发送中断
<i>MDSEND</i>	
<i>SDIO_INT_FLAG_D</i>	数据结束中断
<i>TEND</i>	
<i>SDIO_INT_FLAG_S</i>	起始位错误中断
<i>TBITE</i>	
<i>SDIO_INT_FLAG_D</i>	数据块已发送/已接收中断
<i>TBLKEND</i>	
<i>SDIO_INT_FLAG_C</i>	正在传输命令中断
<i>MDRUN</i>	
<i>SDIO_INT_FLAG_T</i>	正在传输数据中断
<i>XRUN</i>	
<i>SDIO_INT_FLAG_R</i>	正在接收数据中断
<i>XRUN</i>	
<i>SDIO_INT_FLAG_T</i>	发送FIFO半满中断
<i>FH</i>	
<i>SDIO_INT_FLAG_R</i>	接收FIFO半满中断
<i>FH</i>	
<i>SDIO_INT_FLAG_T</i>	发送FIFO满中断
<i>FF</i>	
<i>SDIO_INT_FLAG_R</i>	接收FIFO满中断
<i>FF</i>	

<i>SDIO_INT_FLAG_T</i> <i>FE</i>	发送FIFO空中断
<i>SDIO_INT_FLAG_R</i> <i>FE</i>	接收FIFO空中断
<i>SDIO_INT_FLAG_T</i> <i>XDTVAL</i>	发送FIFO中的数据有效中断
<i>SDIO_INT_FLAG_R</i> <i>XDTVAL</i>	接收FIFO中的数据有效中断
<i>SDIO_INT_FLAG_S</i> <i>DIOINT</i>	SD I/O中断已接收中断
<i>SDIO_INT_FLAG_A</i> <i>TAEND</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get the interrupt flags state of SDIO */
FlagStatus flag_value;

flag_value = sdio_interrupt_flag_get(SDIO_INT_FLAG_DTEND);
```

函数 **sdio_interrupt_flag_clear**

函数sdio_interrupt_flag_clear描述见下表：

表 3-570. 函数 **sdio_interrupt_flag_clear**

函数名称	sdio_interrupt_flag_clear
函数原形	void sdio_interrupt_flag_clear(uint32_t int_flag);
功能描述	清除SDIO的中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
int_flag	SDIO中断标志位状态

<i>SDIO_INT_FLAG_C</i>	命令响应CRC错误中断
<i>SDIO_INT_FLAG_D</i>	数据CRC错误中断
<i>SDIO_INT_FLAG_C</i>	命令响应超时中断
<i>SDIO_INT_FLAG_D</i>	数据超时中断
<i>SDIO_INT_FLAG_T</i>	发送FIFO下溢错误中断
<i>SDIO_INT_FLAG_R</i>	接收FIFO上溢错误中断
<i>SDIO_INT_FLAG_C</i>	命令响应已接收中断
<i>SDIO_INT_FLAG_C</i>	命令已发送中断
<i>SDIO_INT_FLAG_D</i>	数据结束中断
<i>SDIO_INT_FLAG_S</i>	起始位错误中断
<i>SDIO_INT_FLAG_D</i>	数据块已发送/已接收中断
<i>SDIO_INT_FLAG_S</i>	SD I/O中断已接收中断
<i>SDIO_INT_FLAG_A</i>	CE-ATA命令完成信号已接收中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the interrupt pending flags of SDIO */
sdio_interrupt_flag_clear(SDIO_INT_FLAG_DTEND);
```

函数 `sdio_readwait_enable`

函数`sdio_readwait_enable`描述见下表:

表 3-571. 函数 `sdio_readwait_enable`

函数名称	<code>sdio_readwait_enable</code>
函数原形	<code>void sdio_readwait_enable(void);</code>
功能描述	使能读等待模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the read wait mode(SD I/O only) */
sdio_readwait_enable();
```

函数 `sdio_readwait_disable`

函数`sdio_readwait_disable`描述见下表:

表 3-572. 函数 `sdio_readwait_disable`

函数名称	<code>sdio_readwait_disable</code>
函数原形	<code>void sdio_readwait_disable(void);</code>
功能描述	禁能读等待模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable the read wait mode(SD I/O only) */

sdio_readwait_disable();
```

函数 **sdio_stop_readwait_enable**

函数sdio_stop_readwait_enable描述见下表：

表 3-573. 函数 **sdio_stop_readwait_enable**

函数名称	sdio_stop_readwait_enable
函数原形	void sdio_stop_readwait_enable(void);
功能描述	使能停止读等待过程的功能（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the function that stop the read wait process(SD I/O only) */

sdio_stop_readwait_enable();
```

函数 **sdio_stop_readwait_disable**

函数sdio_stop_readwait_disable描述见下表：

表 3-574. 函数 **sdio_stop_readwait_disable**

函数名称	sdio_stop_readwait_disable
函数原形	void sdio_stop_readwait_disable(void);

功能描述	禁能停止读等待过程的功能（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the function that stop the read wait process(SD I/O only) */
sdio_stop_readwait_disable();
```

函数 **sdio_readwait_type_set**

函数sdio_readwait_type_set描述见下表：

表 3-575. 函数 **sdio_readwait_type_set**

函数名称	sdio_readwait_type_set
函数原形	void sdio_readwait_type_set(uint32_t readwait_type);
功能描述	设置读等待类型（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
readwait_type	SD I/O 读等待模式
SDIO_READWAITT_YPE_CLK	通过停止SDIO_CLK控制读等待
SDIO_READWAITT_YPE_DAT2	使用SDIO_DAT[2] 控制读等待
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* set the read wait type(SD I/O only) */
sdio_readwait_type_set(uint32_t readwait_type);
```

函数 **sdio_operation_enable**

函数sdio_operation_enable描述见下表：

表 3-576. 函数 **sdio_operation_enable**

函数名称	sdio_operation_enable
函数原形	void sdio_operation_enable(void);
功能描述	使能SD I/O模式特定操作（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SD I/O mode specific operation(SD I/O only) */
sdio_operation_enable();
```

函数 **sdio_operation_disable**

函数sdio_operation_disable描述见下表：

表 3-577. 函数 **sdio_operation_disable**

函数名称	sdio_operation_disable
函数原形	void sdio_operation_disable(void);
功能描述	禁能SD I/O模式特定操作（仅限SD I/O模式）
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the SD I/O mode specific operation(SD I/O only) */
void sdio_operation_disable();
```

函数 **sdio_suspend_enable**

函数sdio_suspend_enable描述见下表：

表 3-578. 函数 **sdio_suspend_enable**

函数名称	sdio_suspend_enable
函数原形	void sdio_suspend_enable(void);
功能描述	使能SD I/O暂停模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the SD I/O suspend operation(SD I/O only) */
void sdio_suspend_enable();
```

函数 **sdio_suspend_disable**

函数sdio_suspend_disable描述见下表:

表 3-579. 函数 **sdio_suspend_disable**

函数名称	sdio_suspend_disable
函数原形	void sdio_suspend_disable(void);
功能描述	禁能SD I/O暂停模式（仅限SD I/O模式）
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the SD I/O suspend operation(SD I/O only) */

sdio_suspend_disable();
```

函数 **sdio_ceata_command_enable**

函数sdio_ceata_command_enable描述见下表:

表 3-580. 函数 **sdio_ceata_command_enable**

函数名称	sdio_ceata_command_enable
函数原形	void sdio_ceata_command_enable(void);
功能描述	使能CE-ATA命令(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable the CE-ATA command(CE-ATA only) */

sdio_ceata_command_enable();
```

函数 **sdio_ceata_command_disable**

函数sdio_ceata_command_disable描述见下表：

表 3-581. 函数 **sdio_ceata_command_disable**

函数名称	sdio_ceata_command_disable
函数原形	void sdio_ceata_command_disable(void);
功能描述	禁能CE-ATA命令(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CE-ATA command(CE-ATA only) */

sdio_ceata_command_disable();
```

函数 **sdio_ceata_interrupt_enable**

函数sdio_ceata_interrupt_enable描述见下表：

表 3-582. 函数 **sdio_ceata_interrupt_enable**

函数名称	sdio_ceata_interrupt_enable
函数原形	void sdio_ceata_interrupt_enable(void);

功能描述	使能CE-ATA中断(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the CE-ATA interrupt(CE-ATA only) */

sdio_ceata_interrupt_enable();
```

函数 **sdio_ceata_interrupt_disable**

函数sdio_ceata_interrupt_disable描述见下表：

表 3-583. 函数 **sdio_ceata_interrupt_disable**

函数名称	sdio_ceata_interrupt_disable
函数原形	void sdio_ceata_interrupt_disable(void);
功能描述	禁能CE-ATA中断(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CE-ATA interrupt(CE-ATA only) */
```

```
sdio_ceata_interrupt_disable();
```

函数 **sdio_ceata_command_completion_enable**

函数sdio_ceata_command_completion_enable描述见下表:

表 3-584. 函数 sdio_ceata_command_completion_enable

函数名称	sdio_ceata_command_completion_enable
函数原形	void sdio_ceata_command_completion_enable(void);
功能描述	使能CE-ATA命令完成信号(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_enable();
```

函数 **sdio_ceata_command_completion_disable**

函数sdio_ceata_command_completion_disable描述见下表:

表 3-585. 函数 sdio_ceata_command_completion_disable

函数名称	sdio_ceata_command_completion_disable
函数原形	void sdio_ceata_command_completion_disable(void);
功能描述	禁能CE-ATA命令完成信号(仅限CE-ATA模式)
先决条件	-
被调用函数	-
输入参数{in}	
-	-

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the CE-ATA command completion signal(CE-ATA only) */
sdio_ceata_command_completion_disable();
```

3.22. SPI

SPI/I2S模块可以通过SPI协议或I2S音频协议与外部设备进行通信。章节[3.22.1](#)描述了SPI/I2S的寄存器列表，章节[3.22.2](#)对SPI/I2S库函数进行说明。

3.22.1. 外设寄存器说明

SPI/I2S寄存器列表如下表所示：

表 3-586. SPI/I2S 寄存器

寄存器名称	寄存器描述
SPI_CTL0	控制寄存器0
SPI_CTL1	控制寄存器1
SPI_STAT	状态寄存器
SPI_DATA	数据寄存器
SPI_CRCPOLY	CRC多项式寄存器
SPI_RCRC	接收CRC寄存器
SPI_TCRC	发送CRC寄存器
SPI_I2SCTL	I2S控制寄存器
SPI_I2SPSC	I2S时钟分频寄存器
SPI_QCTL	四路SPI控制寄存器

3.22.2. 外设库函数说明

SPI/I2S库函数列表如下表所示：

表 3-587. SPI/I2S 库函数

库函数名称	库函数描述
spi_i2s_deinit	复位外设SPIx/I2Sx
spi_struct_para_init	将SPI结构体中所有参数初始化为默认值
spi_init	初始化外设SPIx
spi_enable	使能外设SPIx
spi_disable	失能外设SPIx
i2s_init	初始化外设I2Sx
i2s_psc_config	配置I2Sx预分频器
i2s_enable	使能外设I2Sx
i2s_disable	失能外设I2Sx
spi_nss_output_enable	使能外设SPIx NSS输出
spi_nss_output_disable	失能外设SPIx NSS输出
spi_nss_internal_high	NSS软件模式下NSS引脚拉高
spi_nss_internal_low	NSS软件模式下NSS引脚拉低
spi_dma_enable	使能外设SPIx的DMA功能
spi_dma_disable	失能外设SPIx的DMA功能
spi_i2s_data_frame_format_config	配置外设SPIx/I2Sx数据帧格式
spi_i2s_data_transmit	发送数据
spi_i2s_data_receive	接收数据
spi_bidirectional_transfer_config	配置外设SPIx的数据传输方向
spi_crc_polynomial_set	设置外设SPIx的CRC多项式值
spi_crc_polynomial_get	获取外设SPIx的CRC多项式值
spi_crc_on	打开外设SPIx的CRC功能
spi_crc_off	关闭外设SPIx的CRC功能
spi_crc_next	设置外设SPIx下一次传输数据为CRC值
spi_crc_get	外设SPIx获取CRC值
spi_ti_mode_enable	使能SPI TI模式
spi_ti_mode_disable	禁能SPI TI模式

库函数名称	库函数描述
spi_nssp_mode_enable	使能SPI NSS脉冲模式
spi_nssp_mode_disable	禁能SPI NSS脉冲模式
qspi_enable	使能四线SPI模式
qspi_disable	禁能四线SPI模式
qspi_write_enable	使能四线SPI写
qspi_read_enable	使能四线SPI读
qspi_io23_output_enable	使能SPI_IO2和SPI_IO3输出
qspi_io23_output_disable	禁能SPI_IO2和SPI_IO3输出
spi_i2s_interrupt_enable	使能外设SPIx/I2Sx中断
spi_i2s_interrupt_disable	失能外设SPIx/I2Sx中断
spi_i2s_interrupt_flag_get	获取外设SPIx/I2Sx中断状态
spi_i2s_flag_get	获取外设SPIx/I2Sx标志状态
spi_crc_error_clear	清除SPIx CRC错误标志状态

结构体 spi_parameter_struct

表 3-588. 结构体 spi_parameter_struct

成员名称	功能描述
device_mode	配置SPI为主机或从机模式 (SPI_MASTER, SPI_SLAVE)
trans_mode	传输模式 (SPI_TRANSMODE_FULLDUPLEX, SPI_TRANSMODE_RECEIVEONLY, SPI_TRANSMODE_BDRECEIVE, SPI_TRANSMODE_BDTRANSMIT)
frame_size	数据帧格式配置 (SPI_FRAMESIZE_16BIT, SPI_FRAMESIZE_8BIT)
nss	配置NSS由软件或硬件控制 (SPI_NSS_SOFT, SPI_NSS_HARD)
endian	大端或小端模式配置 (SPI_ENDIAN_MSB, SPI_ENDIAN_LSB)

clock_polarity_phase	相位和极性配置 (SPI_CK_PL_LOW_PH_1EDGE, SPI_CK_PL_HIGH_PH_1EDGE, SPI_CK_PL_LOW_PH_2EDGE, SPI_CK_PL_HIGH_PH_2EDGE)
prescale	预分频器配置 (SPI_PSC_n (n=2,4,8,16,32,64,128,256))

函数 spi_i2s_deinit

函数spi_i2s_deinit描述见下表:

表 3-589. 函数 spi_i2s_deinit

函数名称	spi_i2s_deinit
函数原形	void spi_i2s_deinit(uint32_t spi_periph);
功能描述	复位外设SPIx/I2Sx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset SPI0 */
spi_i2s_deinit(SPI0);
```

函数 spi_struct_para_init

函数spi_struct_para_init描述见下表:

表 3-590. 函数 spi_struct_para_init

函数名称	spi_struct_para_init
函数原形	void spi_struct_para_init(spi_parameter_struct* spi_struct);

功能描述	将SPI结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
*spi_struct	一个已经定义的spi_parameter_struct结构体变量地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize the parameters of SPI */
spi_parameter_struct spi_init_struct;
spi_struct_para_init(&spi_init_struct);
```

函数 spi_init

函数spi_init描述见下表：

表 3-591. 函数 spi_init

函数名称	spi_init
函数原形	void spi_init(uint32_t spi_periph, spi_parameter_struct* spi_struct);
功能描述	初始化外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
spi_struct	初始化结构体，结构体成员参考 表 3-588. 结构体spi_parameter_struct
输出参数{out}	
-	-

返回值	
-	-

例如：

```

/* initialize SPI0 */

spi_parameter_struct spi_init_struct;

spi_init_struct.trans_mode          = SPI_TRANSMODE_BDTRANSMIT;
spi_init_struct.device_mode         = SPI_MASTER;
spi_init_struct.frame_size         = SPI_FRAMESIZE_8BIT;
spi_init_struct.clock_polarity_phase = SPI_CK_PL_HIGH_PH_2EDGE;
spi_init_struct.nss                 = SPI_NSS_SOFT;
spi_init_struct.prescale            = SPI_PSC_8;
spi_init_struct.endian              = SPI_ENDIAN_MSB;

spi_init(SPI0, &spi_init_struct);

```

函数 spi_enable

函数spi_enable描述见下表：

表 3-592. 函数 spi_enable

函数名称	spi_enable
函数原形	void spi_enable(uint32_t spi_periph);
功能描述	使能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 */
```

```
spi_enable(SPI0);
```

函数 **spi_disable**

函数**spi_disable**描述见下表：

表 3-593. 函数 **spi_disable**

函数名称	spi_disable
函数原形	void spi_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 */
```

```
spi_disable(SPI0);
```

函数 **i2s_init**

函数**i2s_init**描述见下表：

表 3-594. 函数 **i2s_init**

函数名称	i2s_init
函数原形	void i2s_init(uint32_t spi_periph,uint32_t i2s_mode, uint32_t i2s_standard, uint32_t i2s_ckpl);
功能描述	初始化外设I2Sx

先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1,2
输入参数{in}	
i2s_mode	I2S运行模式
I2S_MODE_SLAVE_TX	I2S从机发送模式
I2S_MODE_SLAVE_RX	I2S从机接收模式
I2S_MODE_MASTER_TX	I2S主机发送模式
I2S_MODE_MASTER_RX	I2S主机接收模式
输入参数{in}	
i2s_standard	I2S标准选择
I2S_STD_PHILLIPS	I2S飞利浦标准
I2S_STD_MSB	I2S MSB对齐标准
I2S_STD_LSB	I2S LSB对齐标准
I2S_STD_PCMSHORTFRAME	I2S PCM短帧标准
I2S_STD_PCMLONGFRAME	I2S PCM长帧标准
输入参数{in}	
i2s_ckpol	I2S空闲状态时钟极性
I2S_CKPOL_LOW	I2S_CK空闲状态为低电平
I2S_CKPOL_HIGH	I2S_CK空闲状态为高电平
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* initialize I2S1 */
i2s_init(SPI1, I2S_MODE_MASTERTX, I2S_STD_PHILLIPS, I2S_CKPL_LOW);
```

函数 i2s_psc_config

函数i2s_psc_config描述见下表：

表 3-595. 函数 i2s_psc_config

函数名称	i2s_psc_config
函数原形	void i2s_psc_config(uint32_t spi_periph, uint32_t i2s_audiosample, uint32_t i2s_frameformat, uint32_t i2s_mckout);
功能描述	配置I2Sx预分频器
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
spi_periph	外设I2Sx
SPIx	x=1,2
输入参数{in}	
i2s_audiosample	I2S音频采样频率
I2S_AUDIOSAMPL_E_8K	音频采样频率为8KHz
I2S_AUDIOSAMPL_E_11K	音频采样频率为11KHz
I2S_AUDIOSAMPL_E_16K	音频采样频率为16KHz
I2S_AUDIOSAMPL_E_22K	音频采样频率为22KHz
I2S_AUDIOSAMPL_E_32K	音频采样频率为32KHz
I2S_AUDIOSAMPL_E_44K	音频采样频率为44KHz

<i>E_44K</i>	
<i>I2S_AUDIOSAMPL_E_48K</i>	音频采样频率为48KHz
<i>I2S_AUDIOSAMPL_E_96K</i>	音频采样频率为96KHz
<i>I2S_AUDIOSAMPL_E_192K</i>	音频采样频率为192KHz
输入参数{in}	
<i>i2s_frameformat</i>	I2S数据长度和通道长度
<i>I2S_FRAMEFORMAT_DT16B_CH16B</i>	I2S数据长度为16位，通道长度为16位
<i>I2S_FRAMEFORMAT_DT16B_CH32B</i>	I2S数据长度为16位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT24B_CH32B</i>	I2S数据长度为24位，通道长度为32位
<i>I2S_FRAMEFORMAT_DT32B_CH32B</i>	I2S数据长度为32位，通道长度为32位
输入参数{in}	
<i>i2s_mckout</i>	2S_MCK输出使能
<i>I2S_MCKOUT_ENABLE</i>	I2S_MCK输出使能
<i>I2S_MCKOUT_DISABLE</i>	I2S_MCK输出禁止
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure I2S1 prescaler */

i2s_psc_config(SPI1, I2S_AUDIOSAMPLE_44K, I2S_FRAMEFORMAT_DT16B_CH16B,
I2S_MCKOUT_DISABLE);
  
```

函数 i2s_enable

函数i2s_enable描述见下表:

表 3-596. 函数 i2s_enable

函数名称	i2s_enable
函数原形	void i2s_enable(uint32_t spi_periph);
功能描述	使能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx
SPI/x	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable I2S1 */
```

```
i2s_enable(SPI1);
```

函数 i2s_disable

函数i2s_disable描述见下表:

表 3-597. 函数 i2s_disable

函数名称	i2s_disable
函数原形	void i2s_disable(uint32_t spi_periph);
功能描述	禁能外设I2Sx
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPI/I2Sx

SPIx	x=1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable I2S1 */
i2s_disable(SPI1);
```

函数 **spi_nss_output_enable**

函数spi_nss_output_enable描述见下表：

表 3-598. 函数 **spi_nss_output_enable**

函数名称	spi_nss_output_enable
函数原形	void spi_nss_output_enable(uint32_t spi_periph);
功能描述	使能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS output */
spi_nss_output_enable(SPI0);
```

函数 **spi_nss_output_disable**

函数spi_nss_output_disable描述见下表：

表 3-599. 函数 spi_nss_output_disable

函数名称	spi_nss_output_disable
函数原形	void spi_nss_output_disable(uint32_t spi_periph);
功能描述	禁能外设SPIx NSS输出
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS output */

spi_nss_output_disable(SPI0);
```

函数 spi_nss_internal_high

函数spi_nss_internal_high描述见下表：

表 3-600. 函数 spi_nss_internal_high

函数名称	spi_nss_internal_high
函数原形	void spi_nss_internal_high(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉高
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled high level in software mode */

spi_nss_internal_high(SPI0);
```

函数 **spi_nss_internal_low**

函数spi_nss_internal_low描述见下表：

表 3-601. 函数 **spi_nss_internal_low**

函数名称	spi_nss_internal_low
函数原形	void spi_nss_internal_low(uint32_t spi_periph);
功能描述	NSS软件模式下NSS引脚拉低
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 NSS pin is pulled low level in software mode */

spi_nss_internal_low(SPI0);
```

函数 **spi_dma_enable**

函数spi_dma_enable描述见下表：

表 3-602. 函数 **spi_dma_enable**

函数名称	spi_dma_enable
------	----------------

函数原形	void spi_dma_enable(uint32_t spi_periph, uint8_t dma);
功能描述	使能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit data DMA function */
spi_dma_enable(SPI0, SPI_DMA_TRANSMIT);
```

函数 spi_dma_disable

函数spi_dma_disable描述见下表：

表 3-603. 函数 spi_dma_disable

函数名称	spi_dma_disable
函数原形	void spi_dma_disable(uint32_t spi_periph, uint8_t dma);
功能描述	禁能外设SPIx的DMA功能
先决条件	-
被调用函数	-
输入参数{in}	

spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
dma	SPI DMA模式
SPI_DMA_TRANSMIT	SPI发送缓冲区DMA使能
SPI_DMA_RECEIVE	SPI接收缓冲区DMA使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit data DMA function */
spi_dma_disable(SPI0, SPI_DMA_TRANSMIT);
```

函数 **spi_i2s_data_frame_format_config**

函数spi_i2s_data_frame_format_config描述见下表：

表 3-604. 函数 **spi_i2s_data_frame_format_config**

函数名称	spi_i2s_data_frame_format_config
函数原形	void spi_i2s_data_frame_format_config(uint32_t spi_periph, uint16_t frame_format);
功能描述	配置外设SPIx/I2Sx数据帧格式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
frame_format	SPI帧大小

<i>SPI_FRAME_SIZE_16BIT</i>	SPI 16位数据帧格式
<i>SPI_FRAME_SIZE_8BIT</i>	SPI 8位数据帧格式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure SPI1/I2S1 data frame format size is 16 bits */
spi_i2s_data_frame_format_config(SPI1, SPI_FRAME_SIZE_16BIT);
```

函数 **spi_i2s_data_transmit**

函数spi_i2s_data_transmit描述见下表：

表 3-605. 函数 **spi_i2s_data_transmit**

函数名称	spi_i2s_data_transmit
函数原形	void spi_i2s_data_transmit(uint32_t spi_periph, uint16_t data);
功能描述	SPI发送数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
data	16位数据
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 transmit data */

uint16_t spi0_send_array[] = {0x5050,0XA0A0};

spi_i2s_data_transmit(SPI0, spi0_send_array[0]);
```

函数 **spi_i2s_data_receive**

函数spi_i2s_data_receive描述见下表：

表 3-606. 函数 **spi_i2s_data_receive**

函数名称	spi_i2s_data_receive
函数原形	uint16_t spi_i2s_data_receive(uint32_t spi_periph);
功能描述	SPI接收数据
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	16位数据

例如：

```
/* SPI0 receive data */

uint16_t spi0_receive_data;

spi0_receive_data = spi_i2s_data_receive(SPI0);
```

函数 **spi_bidirectional_transfer_config**

函数spi_bidirectional_transfer_config描述见下表：

表 3-607. 函数 **spi_bidirectional_transfer_config**

函数名称	spi_bidirectional_transfer_config
函数原形	void spi_bidirectional_transfer_config(uint32_t spi_periph, uint32_t transfer_direction);

功能描述	配置外设SPIx的数据传输方向
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
transfer_direction	SPI双向传输输出使能
SPI_BIDIRECTION AL_TRANSMIT	SPI工作在只发送模式
SPI_BIDIRECTION AL_RECEIVE	SPI工作在只接收模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 works in transmit-only mode */
spi_bidirectional_transfer_config(SPI0, SPI_BIDIRECTIONAL_TRANSMIT);
```

函数 spi_crc_polynomial_set

函数spi_crc_polynomial_set描述见下表：

表 3-608. 函数 spi_crc_polynomial_set

函数名称	spi_crc_polynomial_set
函数原形	void spi_crc_polynomial_set(uint32_t spi_periph, uint16_t crc_poly);
功能描述	设置外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx

SPIx	x=0,1,2
输入参数{in}	
crc_poly	CRC多项式值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set SPI0 CRC polynomial */

uint16_t CRC_VALUE = 0x5050;

spi_crc_polynomial_set(SPI0,CRC_VALUE);
```

函数 spi_crc_polynomial_get

函数spi_crc_polynomial_get描述见下表：

表 3-609. 函数 spi_crc_polynomial_get

函数名称	spi_crc_polynomial_get
函数原形	uint16_t spi_crc_polynomial_get(uint32_t spi_periph);
功能描述	获取外设SPIx的CRC多项式值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC多项式值 (0-0xFFFF)

例如：

```
/* get SPI0 CRC polynomial */
```

```

uint16_t crc_val;

crc_val = spi_crc_polynomial_get(SPI0);

```

函数 **spi_crc_on**

函数**spi_crc_on**描述见下表:

表 3-610. 函数 **spi_crc_on**

函数名称	spi_crc_on
函数原形	void spi_crc_on(uint32_t spi_periph);
功能描述	打开外设SPIx的CRC功能
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* turn on SPI0 CRC function */

spi_crc_on(SPI0);

```

函数 **spi_crc_off**

函数**spi_crc_off**描述见下表:

表 3-611. 函数 **spi_crc_off**

函数名称	spi_crc_off
函数原形	void spi_crc_off(uint32_t spi_periph);
功能描述	关闭外设SPIx的CRC功能
先决条件	-
被调用函数	-

输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* turn off SPI0 CRC function */

spi_crc_off(SPI0);
```

函数 **spi_crc_next**

函数**spi_crc_next**描述见下表：

表 3-612. 函数 **spi_crc_next**

函数名称	spi_crc_next
函数原形	void spi_crc_next(uint32_t spi_periph);
功能描述	设置外设SPIx下一次传输数据为CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
<i>SPIx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* SPI0 next data is CRC value */

spi_crc_next(SPI0);
```

函数 **spi_crc_get**

函数spi_crc_get描述见下表:

表 3-613. 函数 **spi_crc_get**

函数名称	spi_crc_get
函数原形	uint16_t spi_crc_get(uint32_t spi_periph, uint8_t crc);
功能描述	外设SPIx获取CRC值
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
crc	SPI crc值
SPI_CRC_TX	获取发送CRC寄存器值
SPI_CRC_RX	获取接收CRC寄存器值
输出参数{out}	
-	-
返回值	
uint16_t	16位CRC值 (0-0xFFFF)

例如:

```
/* get SPI0 CRC send value */

uint16_t crc_val;

crc_val = spi_crc_get(SPI0, SPI_CRC_TX);
```

函数 **spi_ti_mode_enable**

函数spi_ti_mode_enable描述见下表:

表 3-614. 函数 **spi_ti_mode_enable**

函数名称	spi_ti_mode_enable
函数原形	void spi_ti_mode_enable(uint32_t spi_periph);

功能描述	使能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 TI mode */
spi_ti_mode_enable(SPI0);
```

函数 spi_ti_mode_disable

函数spi_ti_mode_disable描述见下表：

表 3-615. 函数 spi_ti_mode_disable

函数名称	spi_ti_mode_disable
函数原形	void spi_ti_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI TI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 TI mode */
spi_ti_mode_disable(SPI0);
```

函数 **spi_nssp_mode_enable**

函数spi_nssp_mode_enable描述见下表：

表 3-616. 函数 **spi_nssp_mode_enable**

函数名称	spi_nssp_mode_enable
函数原形	void spi_nssp_mode_enable(uint32_t spi_periph);
功能描述	使能SPI NSS脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 NSS pulse mode */
spi_nssp_mode_enable(SPI0);
```

函数 **spi_nssp_mode_disable**

函数spi_nssp_mode_disable描述见下表：

表 3-617. 函数 **spi_nssp_mode_disable**

函数名称	spi_nssp_mode_disable
函数原形	void spi_nssp_mode_disable(uint32_t spi_periph);
功能描述	禁能SPI NSS脉冲模式
先决条件	-

被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 NSS pulse mode */
spi_nssp_mode_disable(SPI0);
```

函数 qspi_enable

函数qspi_enable描述见下表：

表 3-618. 函数 qspi_enable

函数名称	qspi_enable
函数原形	void qspi_enable(uint32_t spi_periph);
功能描述	使能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire mode */
```

```
qspi_enable(SPI0);
```

函数 **qspi_disable**

函数qspi_disable描述见下表:

表 3-619. 函数 qspi_disable

函数名称	qspi_disable
函数原形	qspi_disable(uint32_t spi_periph);
功能描述	禁能四线SPI模式
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable SPI0 quad wire mode */
qspi_disable(SPI0);
```

函数 **qspi_write_enable**

函数qspi_write_enable描述见下表:

表 3-620. 函数 qspi_write_enable

函数名称	qspi_write_enable
函数原形	void qspi_write_enable(uint32_t spi_periph);
功能描述	使能四线SPI写
先决条件	-
被调用函数	-
输入参数{in}	

spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire write */
qspi_write_enable(SPI0);
```

函数 qspi_read_enable

函数qspi_read_enable描述见下表：

表 3-621. 函数 qspi_read_enable

函数名称	qspi_read_enable
函数原形	void qspi_read_enable(uint32_t spi_periph);
功能描述	使能四线SPI读
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 quad wire read */
qspi_read_enable(SPI0);
```

函数 `qspi_io23_output_enable`

函数`qspi_io23_output_enable`描述见下表:

表 3-622. 函数 `qspi_io23_output_enable`

函数名称	<code>qspi_io23_output_enable</code>
函数原形	<code>void qspi_io23_output_enable(uint32_t spi_periph);</code>
功能描述	使能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx
<code>SPIx</code>	$x=0$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable SPI0 SPI_IO2 and SPI_IO3 pin output */
qspi_io23_output_enable(SPI0);
```

函数 `qspi_io23_output_disable`

函数`qspi_io23_output_disable`描述见下表:

表 3-623. 函数 `qspi_io23_output_disable`

函数名称	<code>qspi_io23_output_disable</code>
函数原形	<code>void qspi_io23_output_disable(uint32_t spi_periph);</code>
功能描述	禁能SPI_IO2和SPI_IO3输出
先决条件	-
被调用函数	-
输入参数{in}	
<code>spi_periph</code>	外设SPIx

SPIx	x=0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 SPI_IO2 and SPI_IO3 pin output */

qspi_io23_output_disable(SPI0);
```

函数 spi_i2s_interrupt_enable

函数spi_i2s_interrupt_enable描述见下表：

表 3-624. 函数 spi_i2s_interrupt_enable

函数名称	spi_i2s_interrupt_enable
函数原形	void spi_i2s_interrupt_enable(uint32_t spi_periph, uint8_t interrupt);
功能描述	使能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_enable(SPI0, SPI_I2S_INT_TBE);
```

函数 spi_i2s_interrupt_disable

函数spi_i2s_interrupt_disable描述见下表：

表 3-625. 函数 spi_i2s_interrupt_disable

函数名称	spi_i2s_interrupt_disable
函数原形	void spi_i2s_interrupt_disable(uint32_t spi_periph, uint8_t interrupt);
功能描述	禁能外设SPIx/I2Sx中断
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断
SPI_I2S_INT_TBE	发送缓冲区空中断使能
SPI_I2S_INT_RBNE	接收缓冲区非空中断使能
SPI_I2S_INT_ERR	错误中断使能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable SPI0 transmit buffer empty interrupt */
spi_i2s_interrupt_disable(SPI0, SPI_I2S_INT_TBE);
```

函数 spi_i2s_interrupt_flag_get

函数spi_i2s_interrupt_flag_get描述见下表：

表 3-626. 函数 spi_i2s_interrupt_flag_get

函数名称	spi_i2s_interrupt_flag_get
函数原形	FlagStatus spi_i2s_interrupt_flag_get(uint32_t spi_periph, uint8_t interrupt);
功能描述	获取外设SPIx/I2Sx中断状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
interrupt	SPI/I2S中断状态
SPI_I2S_INT_FLAG_TBE	发送缓冲区空中断
SPI_I2S_INT_FLAG_RBNE	接收缓冲区非空中断
SPI_I2S_INT_FLAG_RXORERR	接收过载错误中断
SPI_INT_FLAG_CNFERR	配置错误中断
SPI_INT_FLAG_CRCERR	CRC错误中断
I2S_INT_FLAG_TXURERR	发送欠载错误中断
SPI_I2S_INT_FLAG_FERR	帧错误中断
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty interrupt status */
```

```

if(RESET != spi_i2s_interrupt_flag_get(SPI0, SPI_I2S_INT_FLAG_TBE)){
    while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
    spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
}

```

函数 **spi_i2s_flag_get**

函数spi_i2s_flag_get描述见下表：

表 3-627. 函数 **spi_i2s_flag_get**

函数名称	spi_i2s_flag_get
函数原形	FlagStatus spi_i2s_flag_get(uint32_t spi_periph, uint32_t flag);
功能描述	获取外设SPIx/I2Sx标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输入参数{in}	
flag	SPI/I2S标志状态
SPI_FLAG_TBE	发送缓冲区空标志
SPI_FLAG_RBNE	接收缓冲区非空标志
SPI_FLAG_TRANS	通信进行中标志
SPI_I2S_INT_FLAG_RXORERR	接收过载错误标志
SPI_FLAG_CONFE_RR	配置错误标志
SPI_FLAG_CRCER_R	CRC错误标志
SPI_FLAG_FERR	帧错误标志
I2S_FLAG_TBE	发送缓冲区空标志
I2S_FLAG_RBNE	接收缓冲区非空标志

<i>I2S_FLAG_TRANS</i>	通信进行中标志
<i>I2S_FLAG_RXOVERRR</i>	接收过载错误标志
<i>I2S_FLAG_TXUNDERRR</i>	发送欠载错误标志
<i>I2S_FLAG_CH</i>	通道标志
<i>I2S_FLAG_FERR</i>	帧错误标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET 或 RESET

例如：

```
/* get SPI0 transmit buffer empty flag status */
while(RESET == spi_i2s_flag_get(SPI0, SPI_FLAG_TBE));
spi_i2s_data_transmit(SPI0, spi0_send_array[send_n++]);
```

函数 **spi_crc_error_clear**

函数**spi_crc_error_clear**描述见下表：

表 3-628. 函数 **spi_crc_error_clear**

函数名称	spi_crc_error_clear
函数原形	void spi_crc_error_clear(uint32_t spi_periph);
功能描述	清除SPIx CRC错误标志状态
先决条件	-
被调用函数	-
输入参数{in}	
spi_periph	外设SPIx
SPIx	x=0,1,2
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* clear SPI0 CRC error flag status */
spi_crc_error_clear(SPI0);
```

3.23. TIMER

定时器含有可编程的一个无符号计数器，支持输入捕获和输出比较，分为五种类型：高级定时器(TIMERx, x=0, 7)，通用定时器L0(TIMERx, x=1, 2, 3, 4)，通用定时器L1(TIMERx, x=8, 11)，通用定时器L2(TIMERx, x=9, 10, 12, 13)，基本定时器(TIMERx, x=5, 6)，不同类型的定时器具体功能有所差别。章节[3.23.1](#)描述了TIMER的寄存器列表，章节[3.23.2](#)对TIMER库函数进行说明。

3.23.1. 外设寄存器说明

TIMER寄存器列表如下表所示：

表 3-629.TIMER 寄存器

寄存器名称	寄存器描述
TIMER_CTL0	控制寄存器0
TIMER_CTL1	控制寄存器1
TIMER_SMCFG	从模式配置寄存器
TIMER_DMAINTEN	DMA和中断使能寄存器
TIMER_INTF	中断标志寄存器
TIMER_SWEVG	软件事件产生寄存器
TIMER_CHCTL0	通道控制寄存器0
TIMER_CHCTL1	通道控制寄存器1
TIMER_CHCTL2	通道控制寄存器2
TIMER_CNT	计数器寄存器
TIMER_PSC	预分频寄存器
TIMER_CAR	计数器自动重载寄存器
TIMER_CREP	重复计数寄存器
TIMER_CH0CV	通道0捕获/比较寄存器

寄存器名称	寄存器描述
TIMER_CH1CV	通道1捕获/比较寄存器
TIMER_CH2CV	通道2捕获/比较寄存器
TIMER_CH3CV	通道3捕获/比较寄存器
TIMER_CCHP	互补通道保护寄存器
TIMER_DMACFG	DMA配置寄存器
TIMER_DMATB	DMA发送缓冲区寄存器
TIMER_CFG	配置寄存器

3.23.2. 外设库函数说明

TIMER库函数列表如下表所示：

表 3-630. TIMER 库函数

库函数名称	库函数描述
timer_deinit	复位外设TIMERx
timer_struct_para_init	将TIMER初始化结构体中所有参数初始化为默认值
timer_init	初始化外设TIMERx
timer_enable	使能外设TIMERx
timer_disable	禁能外设TIMERx
timer_auto_reload_shadow_enable	TIMERx自动重载影子使能
timer_auto_reload_shadow_disable	TIMERx自动重载影子禁能
timer_update_event_enable	TIMERx更新使能
timer_update_event_disable	TIMERx更新禁能
timer_counter_alignment	设置外设TIMERx的对齐模式
timer_counter_up_direction	设置外设TIMERx向上计数
timer_counter_down_direction	设置外设TIMERx向下计数
timer_prescaler_config	配置外设TIMERx预分频器
timer_repetition_value_config	配置外设TIMERx的重复计数器
timer_autoreload_value_config	配置外设TIMERx的自动重载寄存器
timer_counter_value_config	配置外设TIMERx的计数器值

库函数名称	库函数描述
timer_counter_read	读取外设TIMERx的计数器值
timer_prescaler_read	读取外设TIMERx的预分频器值
timer_single_pulse_mode_config	配置外设TIMERx的单脉冲模式
timer_update_source_config	配置外设TIMERx的更新源
timer_dma_enable	外设TIMERx的DMA使能
timer_dma_disable	外设TIMERx的DMA禁能
timer_channel_dma_request_source_select	外设TIMERx的通道DMA请求源选择
timer_dma_transfer_config	配置外设TIMERx的DMA模式
timer_event_software_generate	软件产生事件
timer_break_struct_para_init	将TIMER中止功能参数结构体中所有参数初始化为默认值
timer_break_config	配置中止功能
timer_break_enable	使能TIMERx的中止功能
timer_break_disable	禁能TIMERx的中止功能
timer_automatic_output_enable	自动输出使能
timer_automatic_output_disable	自动输出禁能
timer_primary_output_config	所有的通道输出使能
timer_channel_control_shadow_config	通道换相控制影子寄存器配置
timer_channel_control_shadow_update_config	通道换相控制影子寄存器更新控制
timer_channel_output_struct_para_init	将TIMER通道输出参数结构体中所有参数初始化为默认值
timer_channel_output_config	外设TIMERx的通道输出配置
timer_channel_output_mode_config	配置外设TIMERx通道输出比较模式
timer_channel_output_pulse_value_config	配置外设TIMERx的通道输出比较值
timer_channel_output_shadow_config	配置TIMERx通道输出比较影子寄存器功能

库函数名称	库函数描述
timer_channel_output_fast_config	配置TIMERx通道输出比较快速功能
timer_channel_output_clear_config	配置TIMERx的通道输出比较清0功能
timer_channel_output_polarity_config	通道输出极性配置
timer_channel_complementary_output_polarity_config	互补通道输出极性配置
timer_channel_output_state_config	配置通道状态
timer_channel_complementary_output_state_config	配置互补通道输出状态
timer_channel_input_struct_para_init	将TIMER通道输入参数结构体中所有参数初始化为默认值
timer_input_capture_config	配置TIMERx输入捕获参数
timer_channel_input_capture_prescaler_config	配置TIMERx通道输入捕获预分频值
timer_channel_capture_value_register_read	读取通道输入捕获值
timer_input_pwm_capture_config	配置TIMERx捕获PWM输入参数
timer_hall_mode_config	配置TIMERx的HALL接口功能
timer_input_trigger_source_select	TIMERx的输入触发源选择
timer_master_output_trigger_source_select	选择TIMERx主模式输出触发
timer_slave_mode_select	TIMERx从模式配置
timer_master_slave_mode_config	TIMERx主从模式配置
timer_external_trigger_config	配置TIMERx外部触发输入
timer_quadrature_decoder_mode_config	TIMERx配置为编码器模式
timer_internal_clock_config	TIMERx配置为内部时钟模式
timer_internal_trigger_as_external_clock_config	配置TIMERx的内部触发为时钟源
timer_external_trigger_as_external_clock_config	配置TIMERx的外部触发作为时钟源

库函数名称	库函数描述
timer_external_clock_mode0_config	配置TIMERx外部时钟模式0, ETI作为时钟源
timer_external_clock_mode1_config	配置TIMERx外部时钟模式1
timer_external_clock_mode1_disable	TIMERx外部时钟模式1禁能
timer_write_chxval_register_config	配置TIMERx写CHxVAL选择位
timer_output_value_selection_config	配置TIMERx输出值选择位
timer_interrupt_enable	外设TIMERx中断使能
timer_interrupt_disable	外设TIMERx中断禁能
timer_interrupt_flag_get	获取外设TIMERx中断标志
timer_interrupt_flag_clear	清除外设TIMERx的中断标志
timer_flag_get	获取外设TIMERx的状态标志
timer_flag_clear	清除外设TIMERx状态标志

结构体 timer_parameter_struct

表 3-631. 结构体 timer_parameter_struct

成员名称	功能描述
prescaler	预分频值 (0~65535)
alignedmode	对齐模式 (TIMER_COUNTER_EDGE, TIMER_COUNTER_CENTER_DOWN, TIMER_COUNTER_CENTER_UP, TIMER_COUNTER_CENTER_BOTH)
counterdirection	计数方向 (TIMER_COUNTER_UP, TIMER_COUNTER_DOWN)
period	周期 (0~65535)
clockdivision	时钟分频因子 (TIMER_CKDIV_DIV1, TIMER_CKDIV_DIV2, TIMER_CKDIV_DIV4)
repetitioncounter	重复计数器值 (0~255)

结构体 timer_break_parameter_struct

表 3-632. 结构体 timer_break_parameter_struct

成员名称	功能描述
runoffstate	运行模式下“关闭状态”配置 (TIMER_ROS_STATE_ENABLE, TIMER_ROS_STATE_DISABLE)

成员名称	功能描述
ideloffstate	空闲模式下“关闭状态”配置 (TIMER_IOS_STATE_ENABLE, TIMER_IOS_STATE_DISABLE)
deadtime	死区时间 (0~255)
breakpolarity	中止信号极性 (TIMER_BREAK_POLARITY_LOW, TIMER_BREAK_POLARITY_HIGH)
outputautostate	自动输出使能 (TIMER_OUTAUTO_ENABLE, TIMER_OUTAUTO_DISABLE)
protectmode	互补寄存器保护控制 (TIMER_CCHP_PROT_OFF, TIMER_CCHP_PROT_0, TIMER_CCHP_PROT_1, TIMER_CCHP_PROT_2)
breakstate	中止使能 (TIMER_BREAK_ENABLE, TIMER_BREAK_DISABLE)

结构体 timer_oc_parameter_struct

表 3-633. 结构体 timer_oc_parameter_struct

成员名称	功能描述
outputstate	通道输出状态 (TIMER_CCX_ENABLE, TIMER_CCX_DISABLE)
outputnstate	互补通道输出状态 (TIMER_CCXN_ENABLE, TIMER_CCXN_DISABLE)
ocpolarity	通道输出极性 (TIMER_OC_POLARITY_HIGH, TIMER_OC_POLARITY_LOW)
ocnpolarity	互补通道输出极性 (TIMER_OCN_POLARITY_HIGH, TIMER_OCN_POLARITY_LOW)
ocidlestate	空闲状态下通道输出 (TIMER_OC_IDLE_STATE_LOW, TIMER_OC_IDLE_STATE_HIGH)
ocnidlestate	空闲状态下互补通道输出 (TIMER_OCN_IDLE_STATE_LOW, TIMER_OCN_IDLE_STATE_HIGH)

结构体 timer_ic_parameter_struct

表 3-634. 结构体 timer_ic_parameter_struct

成员名称	功能描述
icpolarity	通道输入极性 (TIMER_IC_POLARITY_RISING, TIMER_IC_POLARITY_FALLING, TIMER_IC_POLARITY_BOTH_EDGE)
icselection	通道输入模式选择 (TIMER_IC_SELECTION_DIRECTTI, TIMER_IC_SELECTION_INDIRECTTI, TIMER_IC_SELECTION_ITS)
icprescaler	通道输入捕获预分频 (TIMER_IC_PSC_DIV1, TIMER_IC_PSC_DIV2)

成员名称	功能描述
	TIMER_IC_PSC_DIV4, TIMER_IC_PSC_DIV8)
icfilter	通道输入捕获滤波 (0~15)

函数 timer_deinit

函数timer_deinit描述见下表:

表 3-635. 函数 timer_deinit

函数名称	timer_deinit
函数原型	void timer_deinit(uint32_t timer_periph);
功能描述	复位外设TIMERx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset TIMER0 */
timer_deinit (TIMER0);
```

函数 timer_struct_para_init

函数timer_struct_para_init描述见下表:

表 3-636. Function timer_struct_para_init

函数名称	timer_struct_para_init
函数原型	void timer_struct_para_init(timer_parameter_struct* initpara);
功能描述	将TIMER初始化参数结构体中所有参数初始化为默认值
先决条件	-

被调用函数	-
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 表 3-631. 结构体 timer_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER init parameter struct with a default value */

timer_parameter_struct timer_initpara;

timer_struct_para_init(timer_initpara);
```

函数 timer_init

函数timer_init描述见下表：

表 3-637. 函数 timer_init

函数名称	timer_init
函数原型	void timer_init(uint32_t timer_periph, timer_parameter_struct* initpara);
功能描述	初始化外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
initpara	TIMER初始化结构体，结构体成员参考 表 3-631. 结构体 timer_parameter_struct 。
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* initialize TIMER0 */

timer_parameter_struct timer_initpara;

timer_initpara.prescaler      = 107;
timer_initpara.alignedmode    = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection = TIMER_COUNTER_UP;
timer_initpara.period         = 999;
timer_initpara.clockdivision  = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 1;

timer_init(TIMER0, &timer_initpara);
```

函数 **timer_enable**

函数timer_enable描述见下表：

表 3-638. 函数 **timer_enable**

函数名称	timer_enable
函数原型	void timer_enable(uint32_t timer_periph);
功能描述	使能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 */
```

```
timer_enable (TIMER0);
```

函数 **timer_disable**

函数timer_disable描述见下表:

表 3-639. 函数 timer_disable

函数名称	timer_disable
函数原型	void timer_disable(uint32_t timer_periph);
功能描述	禁能外设TIMERx
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable TIMER0 */
timer_disable (TIMER0);
```

函数 **timer_auto_reload_shadow_enable**

函数timer_auto_reload_shadow_enable描述见下表:

表 3-640. 函数 timer_auto_reload_shadow_enable

函数名称	timer_auto_reload_shadow_enable
函数原型	void timer_auto_reload_shadow_enable(uint32_t timer_periph);
功能描述	TIMERx自动重载影子使能
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx(x=0..13)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_enable (TIMER0);
```

函数 **timer_auto_reload_shadow_disable**

函数**timer_auto_reload_shadow_disable**描述见下表：

表 3-641. 函数 **timer_auto_reload_shadow_disable**

函数名称	timer_auto_reload_shadow_disable
函数原型	void timer_auto_reload_shadow_disable (uint32_t timer_periph);
功能描述	TIMERx自动重载影子禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..13)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 auto reload shadow function */
timer_auto_reload_shadow_disable (TIMER0);
```

函数 timer_update_event_enable

函数timer_update_event_enable描述见下表:

表 3-642. 函数 timer_update_event_enable

函数名称	timer_update_event_enable
函数原型	void timer_update_event_enable(uint32_t timer_periph);
功能描述	TIMERx更新使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable TIMER0 the update event */
```

```
timer_update_event_enable (TIMER0);
```

函数 timer_update_event_disable

函数timer_update_event_disable描述见下表:

表 3-643. 函数 timer_update_event_disable

函数名称	timer_update_event_disable
函数原型	void timer_update_event_disable (uint32_t timer_periph);
功能描述	TIMERx更新禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设

<i>TIMERx(x=0..13)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the update event */

timer_update_event_disable (TIMER0);
```

函数 timer_counter_alignment

函数timer_counter_alignment描述见下表：

表 3-644. 函数 timer_counter_alignment

函数名称	timer_counter_alignment
函数原型	void timer_counter_alignment(uint32_t timer_periph, uint16_t aligned);
功能描述	设置外设TIMERx的对齐模式
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输入参数{in}	
<i>aligned</i>	对齐模式
<i>TIMER_COUNTER_EDGE</i>	无中央对齐计数模式(边沿对齐模式), DIR位指定了计数方向
<i>TIMER_COUNTER_CENTER_DOWN</i>	中央对齐向下计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模式 (TIMERx_CHCTL0寄存器中CHxMS=00), 只有在向下计数时, 通道的比较中断标志置1
<i>TIMER_COUNTER_CENTER_UP</i>	中央对齐向上计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模式 (TIMERx_CHCTL0寄存器中CHxMS=00), 只有在向上计数时, 通道的比较中断标志置1
<i>TIMER_COUNTER</i>	中央对齐上下计数置1模式。计数器在中央计数模式计数, 通道被配置在输出模

<code>_CENTER_BOTH</code>	式 (TIMERx_CHCTL0寄存器中CHxMS=00)，在向上和向下计数时，通道的比较中断标志都会置1
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter center-aligned and counting up assert mode */
timer_counter_alignment(TIMER0, TIMER_COUNTER_CENTER_UP);
```

函数 `timer_counter_up_direction`

函数`timer_counter_up_direction`描述见下表：

表 3-645. 函数 `timer_counter_up_direction`

函数名称	timer_counter_up_direction
函数原型	void timer_counter_up_direction(uint32_t timer_periph);
功能描述	设置外设TIMERx向上计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
<code>timer_periph</code>	TIMER外设
<code>TIMERx(x=0..4,7)</code>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set TIMER0 counter up direction */
timer_counter_up_direction(TIMER0);
```

函数 timer_counter_down_direction

函数timer_counter_down_direction描述见下表:

表 3-646. 函数 timer_counter_down_direction

函数名称	timer_counter_down_direction
函数原型	void timer_counter_down_direction(uint32_t timer_periph);
功能描述	设置外设TIMERx向下计数
先决条件	计数器设置为无中央对齐计数模式（边沿对齐模式）
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* set TIMER0 counter down direction */
timer_counter_down_direction (TIMER0);
```

函数 timer_prescaler_config

函数timer_prescaler_config描述见下表:

表 3-647. 函数 timer_prescaler_config

函数名称	timer_prescaler_config
函数原型	void timer_prescaler_config(uint32_t timer_periph, uint16_t prescaler, uint8_t pscreload);
功能描述	配置外设TIMERx预分频器
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx(x=0..13)</i>	TIMER外设选择
输入参数{in}	
prescaler	预分频值, 0~65535
输入参数{in}	
pscreload	预分频值加载模式
<i>TIMER_PSC_RELOAD_NOW</i>	预分频值立即加载
<i>TIMER_PSC_RELOAD_UPDATE</i>	预分频值在下次更新事件发生时加载
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 prescaler */
timer_prescaler_config(TIMER0, 3000, TIMER_PSC_RELOAD_NOW);
```

函数 **timer_repetition_value_config**

函数**timer_repetition_value_config**描述见下表：

表 3-648. 函数 **timer_repetition_value_config**

函数名称	timer_repetition_value_config
函数原型	void timer_repetition_value_config(uint32_t timer_periph, uint16_t repetition);
功能描述	配置外设TIMERx的重复计数器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	TIMER外设选择
输入参数{in}	

repetition	重复计数器值, 取值范围0~255
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 repetition register value */
timer_repetition_value_config (TIMER0, 98);
```

函数 **timer_autoreload_value_config**

函数timer_autoreload_value_config描述见下表:

表 3-649. 函数 **timer_autoreload_value_config**

函数名称	timer_autoreload_value_config
函数原型	void timer_autoreload_value_config(uint32_t timer_periph, uint16_t autoreload);
功能描述	配置外设TIMERx的自动重载寄存器
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..13)</i>	TIMER外设选择
输入参数{in}	
autoreload	计数器自动重载值 (0-0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER autoreload register value */
timer_autoreload_value_config (TIMER0, 3000);
```

函数 timer_counter_value_config

函数timer_counter_value_config描述见下表:

表 3-650. 函数 timer_counter_value_config

函数名称	timer_counter_value_config
函数原型	void timer_counter_value_config(uint32_t timer_periph, uint16_t counter);
功能描述	配置外设TIMERx的计数器值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
counter	计数器值 (0-0xFFFF)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 counter register value */
timer_counter_value_config (TIMER0);
```

函数 timer_counter_read

函数timer_counter_read描述见下表:

表 3-651. 函数 timer_counter_read

函数名称	timer_counter_read
函数原型	uint32_t timer_counter_read(uint32_t timer_periph);
功能描述	读取外设TIMERx的计数器值
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint32_t	外设TIMERx的计数器值 (0x0000~0xFFFF)

例如：

```
/* read TIMER0 counter value */

uint32_t i = 0;

i = timer_counter_read(TIMER0);
```

函数 timer_prescaler_read

函数timer_prescaler_read描述见下表：

表 3-652. 函数 timer_prescaler_read

输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输出参数{out}	
-	-
返回值	
uint16_t	外设TIMERx的预分频器值 (0x0000~0xFFFF)

例如：

```
/* read TIMER0 prescaler value */
```

```

uint16_t i = 0;
i = timer_prescaler_read(TIMER0);

```

函数 timer_single_pulse_mode_config

函数timer_single_pulse_mode_config描述见下表：

表 3-653. 函数 timer_single_pulse_mode_config

函数名称	timer_single_pulse_mode_config
函数原型	void timer_single_pulse_mode_config(uint32_t timer_periph, uint8_t spmode);
功能描述	配置外设TIMERx的单脉冲模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..8,11)	TIMER外设选择
输入参数{in}	
spmode	脉冲模式
TIMER_SP_MODE_SINGLE	单脉冲模式计数
TIMER_SP_MODE_REPETITIVE	重复模式计数
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure TIMER0 single pulse mode */
timer_single_pulse_mode_config (TIMER0, TIMER_SP_MODE_SINGLE);

```

函数 timer_update_source_config

函数timer_update_source_config描述见下表：

表 3-654. 函数 timer_update_source_config

函数名称	timer_update_source_config
函数原型	void timer_update_source_config(uint32_t timer_periph, uint32_t update);
功能描述	配置外设TIMERx的更新源
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..13)	TIMER外设选择
输入参数{in}	
update	更新源
TIMER_UPDATE_SRC_GLOBAL	下述任一事件产生更新中断或DMA请求： – UPG位被置1 – 计数器溢出/下溢 – 从模式控制器产生的更新
TIMER_UPDATE_SRC_REGULAR	只有计数器溢出/下溢才产生更新中断或DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER update only by counter overflow/underflow */
timer_update_source_config (TIMER0, TIMER_UPDATE_SRC_REGULAR);
```

函数 timer_dma_enable

函数timer_dma_enable描述见下表：

表 3-655. 函数 timer_dma_enable

函数名称	timer_dma_enable
函数原型	void timer_dma_enable(uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA使能

先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0..7)
TIMER_DMA_CH0_D	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH1_D	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH2_D	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH3_D	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CMT_D	换相DMA更新请求, TIMERx(x=0,7)
TIMER_DMA_TRG_D	触发DMA请求使能, TIMERx(x=0..4,7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable the TIMER0 update DMA */
timer_dma_enable (TIMER0, TIMER_DMA_UPD);
```

函数 **timer_dma_disable**

函数**timer_dma_disable**描述见下表：

表 3-656. 函数 timer_dma_disable

函数名称	timer_dma_disable
函数原型	void timer_dma_disable (uint32_t timer_periph, uint16_t dma);
功能描述	外设TIMERx的DMA禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma	DMA源
TIMER_DMA_UPD	更新DMA请求, TIMERx(x=0..7)
TIMER_DMA_CH0_D	通道0比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH1_D	通道1比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH2_D	通道2比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CH3_D	通道3比较/捕获 DMA请求, TIMERx(x=0..4,7)
TIMER_DMA_CMT_D	换相DMA更新请求, TIMERx(x=0,7)
TIMER_DMA_TRG_D	触发DMA请求使能, TIMERx(x=0..4,7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable the TIMER0 update DMA */
timer_dma_disable (TIMER0, TIMER_DMA_UPD);
```

函数 timer_channel_dma_request_source_select

函数timer_channel_dma_request_source_select描述见下表:

表 3-657. 函数 timer_channel_dma_request_source_select

函数名称	timer_channel_dma_request_source_select
函数原型	void timer_channel_dma_request_source_select(uint32_t timer_periph, uint32_t dma_request);
功能描述	外设TIMERx的通道DMA请求源选择
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
dma_request	通道的DMA请求源选择
TIMER_DMAREQUEST_CHANNELEVENT	当通道捕获/比较事件发生时，发送通道n的DMA请求
TIMER_DMAREQUEST_UPDATEEVENT	当更新事件发生，发送通道n的DMA请求
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* TIMER0 channel DMA request of channel n is sent when channel y event occurs */

timer_channel_dma_request_source_select (TIMER0,
TIMER_DMAREQUEST_CHANNELEVENT);
```

函数 timer_dma_transfer_config

函数timer_dma_transfer_config描述见下表:

表 3-658. 函数 timer_dma_transfer_config

函数名称	timer_dma_transfer_config
函数原型	void timer_dma_transfer_config(uint32_t timer_periph, uint32_t dma_baseaddr, uint32_t dma_lenth);
功能描述	配置外设TIMERx的DMA模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
dma_baseaddr	DMA传输起始地址
TIMER_DMACFG_DMATA_CTL0	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL0, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CTL1	DMA传输起始地址: TIMER_DMACFG_DMATA_CTL1, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_SMCFG	DMA传输起始地址: TIMER_DMACFG_DMATA_SMCFG, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_DMAINTEN_N	DMA传输起始地址: TIMER_DMACFG_DMATA_DMAINTEN, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_INTF	DMA传输起始地址: TIMER_DMACFG_DMATA_INTF, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_SWEVG	DMA传输起始地址: TIMER_DMACFG_DMATA_SWEVG, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL0	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL0, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL1	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL1, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CHCTL2	DMA传输起始地址: TIMER_DMACFG_DMATA_CHCTL2, TIMERx(x=0..4,7)
TIMER_DMACFG_DMATA_CNT	DMA传输起始地址: TIMER_DMACFG_DMATA_CNT, TIMERx(x=0..4,7)

<i>TIMER_DMACFG_DMATA_PSC</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_PSC, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CAR</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CAR, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CREP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CREP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_CH0CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH0CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CH1CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH1CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CH2CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH2CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CH3CV</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CH3CV, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_CCHP</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_CCHP, TIMERx(x=0,7)
<i>TIMER_DMACFG_DMATA_DMACFG</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMACFG, TIMERx(x=0..4,7)
<i>TIMER_DMACFG_DMATA_DMATB</i>	DMA传输起始地址: TIMER_DMACFG_DMATA_DMATB, TIMERx(x=0..4,7)
输入参数{in}	
dma_lenth	DMA传输长度
<i>TIMER_DMACFG_DMATC_xTRANSFER</i>	x=1..18, DMA传输 x 次
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure the TIMER0 DMA transfer */

timer_dma_transfer_config (TIMER0, TIMER_DMACFG_DMATA_CTL0,
  TIMER_DMACFG_DMATC_5TRANSFER);
```

函数 timer_event_software_generate

函数timer_event_software_generate描述见下表：

表 3-659. 函数 timer_event_software_generate

函数名称	timer_event_software_generate
函数原型	void timer_event_software_generate(uint32_t timer_periph, uint16_t event);
功能描述	软件产生事件
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
event	事件源
TIMER_EVENT_SR_C_UPG	更新事件产生, TIMERx(x=0..13)
TIMER_EVENT_SR_C_CH0G	通道0捕获或比较事件发生, TIMERx(x=0..4,7..13)
TIMER_EVENT_SR_C_CH1G	通道1捕获或比较事件发生, TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SR_C_CH2G	通道2捕获或比较事件发生, TIMERx(x=0..4,7)
TIMER_EVENT_SR_C_CH3G	通道3捕获或比较事件发生, TIMERx(x=0..4,7)
TIMER_EVENT_SR_C_CMTG	通道换相更新事件发生, TIMERx(x=0,7)
TIMER_EVENT_SR_C_TRGG	触发事件产生, TIMERx(x=0..4,7,8,11)
TIMER_EVENT_SR_C_BRKG	产生中止事件, TIMERx(x=0,7)
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* software generate update event*/
timer_event_software_generate (TIMER0, TIMER_EVENT_SRC_UPG);
```

函数 timer_break_struct_para_init

函数timer_break_struct_para_init描述见下表：

表 3-660. 函数 timer_break_struct_para_init

函数名称	timer_break_struct_para_init
函数原型	void timer_break_struct_para_init(timer_break_parameter_struct* breakpara);
功能描述	将TIMER中止功能参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
breakpara	中止功能配置结构体，详见 表 3-632. 结构体 timer_break_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* initialize TIMER break parameter struct with a default value */
timer_break_parameter_struct timer_breakpara;
timer_break_struct_para_init(timer_breakpara);
```

函数 timer_break_config

函数timer_break_config描述见下表：

表 3-661. 函数 timer_break_config

函数名称	timer_break_config
------	--------------------

函数原型	void timer_break_config(uint32_t timer_periph, timer_break_parameter_struct* breakpara);
功能描述	配置中止功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
breakpara	中止功能配置结构体, 详见 表 3-632. 结构体 timer_break_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```

/* configure TIMER0 break function */

timer_break_parameter_struct timer_breakpara;

timer_breakpara.runoffstate      = TIMER_ROS_STATE_DISABLE;
timer_breakpara.idloffstate     = TIMER_IOS_STATE_DISABLE ;
timer_breakpara.deadtime        = 255;
timer_breakpara.breakpolarity   = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode    = TIMER_CCHP_PROT_0;
timer_breakpara.breakstate     = TIMER_BREAK_ENABLE;

timer_break_config(TIMER0,&timer_breakpara);

```

函数 **timer_break_enable**

函数**timer_break_enable**描述见下表:

表 3-662. 函数 timer_break_enable

函数名称	timer_break_enable
函数原型	void timer_break_enable(uint32_t timer_periph);
功能描述	使能TIMERx的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 break function*/
timer_break_enable (TIMER0);
```

函数 timer_break_disable

函数timer_break_disable描述见下表：

表 3-663. 函数 timer_break_disable

函数名称	timer_break_disable
函数原型	void timer_break_disable (uint32_t timer_periph);
功能描述	禁能TIMERx的中止功能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable TIMER0 break function*/
timer_break_disable (TIMER0);
```

函数 **timer_automatic_output_enable**

函数timer_automatic_output_enable描述见下表：

表 3-664. 函数 **timer_automatic_output_enable**

函数名称	timer_automatic_output_enable
函数原型	void timer_automatic_output_enable(uint32_t timer_periph);
功能描述	自动输出使能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] =00 时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 output automatic function */
timer_automatic_output_enable (TIMER0);
```

函数 **timer_automatic_output_disable**

函数timer_automatic_output_disable描述见下表：

表 3-665. 函数 **timer_automatic_output_disable**

函数名称	timer_automatic_output_disable
------	--------------------------------

函数原型	void timer_automatic_output_disable (uint32_t timer_periph);
功能描述	自动输出禁能
先决条件	只有在TIMERx_CCHP寄存器的PROT [1:0] = 00时，才可修改
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 output automatic function */

timer_automatic_output_disable (TIMER0);
```

函数 timer_primary_output_config

函数timer_primary_output_config描述见下表：

表 3-666. 函数 timer_primary_output_config

函数名称	timer_primary_output_config
函数原型	void timer_primary_output_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	所有的通道输出使能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
newvalue	控制状态

ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable TIMER0 primary output function */

timer_primary_output_config (TIMER0, ENABLE);
```

函数 `timer_channel_control_shadow_config`

函数`timer_channel_control_shadow_config`描述见下表：

表 3-667. 函数 `timer_channel_control_shadow_config`

函数名称	timer_channel_control_shadow_config
函数原型	void timer_channel_control_shadow_config(uint32_t timer_periph, ControlStatus newvalue);
功能描述	通道换相控制影子配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
newvalue	控制状态
ENABLE	使能
DISABLE	禁能
输出参数{out}	
-	-
返回值	

-	-
---	---

例如：

```
/* channel capture/compare control shadow register enable */
timer_channel_control_shadow_config(TIMER0, ENABLE);
```

函数 **timer_channel_control_shadow_update_config**

函数timer_channel_control_shadow_update_config描述见下表：

表 3-668. 函数 **timer_channel_control_shadow_update_config**

函数名称	timer_channel_control_shadow_update_config
函数原型	void timer_channel_control_shadow_update_config(uint32_t timer_periph, uint8_t ccuctl);
功能描述	通道换相控制影子寄存器更新控制
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
ccuctl	通道换相控制影子寄存器更新控制
TIMER_UPDATECTL_CCU	CMTG位被置1时更新影子寄存器
TIMER_UPDATECTL_CCUTRI	当CMTG位被置1或检测到TRIGI上升沿时，影子寄存器更新
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel control shadow register update when CMTG bit is set */
timer_channel_control_shadow_update_config(TIMER0, TIMER_UPDATECTL_CCU);
```

函数 timer_channel_output_struct_para_init

函数timer_channel_output_struct_para_init描述见下表:

表 3-669. 函数 timer_channel_output_struct_para_init

函数名称	timer_channel_output_struct_para_init
函数原型	void timer_channel_output_struct_para_init(timer_oc_parameter_struct* ocpara);
功能描述	将TIMER通道输出参数结构体中所有参数初始化为默认值
先决条件	-
被调用函数	-
输入参数{in}	
ocpara	输出通道结构体, 详见 表 3-633. 结构体timer_oc_parameter_struct .
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER channel output parameter struct with a default value */
timer_oc_parameter_struct timer_ocinitpara;
timer_channel_output_struct_para_init(timer_ocinitpara);
```

函数 timer_channel_output_config

函数timer_channel_output_config描述见下表:

表 3-670. 函数 timer_channel_output_config

函数名称	timer_channel_output_config
函数原型	void timer_channel_output_config(uint32_t timer_periph, uint16_t channel, timer_oc_parameter_struct* ocpara);
功能描述	外设TIMERx的通道输出配置
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx(x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx(x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx(x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx(x=0..4,7)
输入参数{in}	
ocpara	输出通道结构体, 详见 表 3-633. 结构体timer_oc_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 channel 0 output function */

timer_oc_parameter_struct timer_ocintpara;

timer_ocintpara.outputstate = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity = TIMER_OC_POLARITY_HIGH;
timer_ocintpara.ocnpolarity = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate = TIMER_OC_IDLE_STATE_HIGH;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;
timer_channel_output_config(TIMER0, TIMER_CH_0, &timer_ocintpara);
```

函数 **timer_channel_output_mode_config**

函数**timer_channel_output_mode_config**描述见下表:

表 3-671. 函数 timer_channel_output_mode_config

函数名称	timer_channel_output_mode_config
------	----------------------------------

函数原型	void timer_channel_output_mode_config(uint32_t timer_periph, uint16_t channel, uint16_t ocmode);
功能描述	配置外设TIMERx通道输出比较模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2, TIMERx (x=0..4,7)
TIMER_CH_3	通道3, TIMERx (x=0..4,7)
输入参数{in}	
ocmode	通道输出比较模式
TIMER_OC_MODE_TIMING	冻结模式
TIMER_OC_MODE_ACTIVE	匹配时设置为高
TIMER_OC_MODE_INACTIVE	匹配时设置为低
TIMER_OC_MODE_TOGGLE	匹配时翻转
TIMER_OC_MODE_LOW	强制为低
TIMER_OC_MODE_HIGH	强制为高
TIMER_OC_MODE_PWM0	PWM模式0
TIMER_OC_MODE	PWM模式1

_PWM1	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel PWM 0 mode */
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
```

函数 timer_channel_output_pulse_value_config

函数timer_channel_output_pulse_value_config描述见下表：

表 3-672. 函数 timer_channel_output_pulse_value_config

函数名称	timer_channel_output_pulse_value_config
函数原型	void timer_channel_output_pulse_value_config(uint32_t timer_periph, uint16_t channel, uint32_t pulse);
功能描述	配置外设TIMERx的通道输出比较值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2, TIMERx (x=0..4,7)
TIMER_CH_3	通道3, TIMERx (x=0..4,7)
输入参数{in}	
pulse	通道输出比较值 (0~65535)
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output pulse value */
timer_channel_output_pulse_value_config(TIMER0, TIMER_CH_0, 399);
```

函数 timer_channel_output_shadow_config

函数timer_channel_output_shadow_config描述见下表：

表 3-673. 函数 timer_channel_output_shadow_config

函数名称	timer_channel_output_shadow_config
函数原型	void timer_channel_output_shadow_config(uint32_t timer_periph, uint16_t channel, uint16_t ocshadow);
功能描述	配置TIMERx通道输出比较影子寄存器功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2, TIMERx (x=0..4,7)
TIMER_CH_3	通道3, TIMERx (x=0..4,7)
输入参数{in}	
ocshadow	输出比较影子寄存器功能状态
TIMER_OC_SHAD_OW_ENABLE	使能输出比较影子寄存器
TIMER_OC_SHAD	禁能输出比较影子寄存器

<i>OW_DISABLE</i>	
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/*configure TIMER0 channel 0 output shadow function */
timer_channel_output_shadow_config (TIMER0, TIMER_CH_0,
TIMER_OC_SHADOW_ENABLE);
```

函数 **timer_channel_output_fast_config**

函数**timer_channel_output_fast_config**描述见下表：

表 3-674. 函数 **timer_channel_output_fast_config**

函数名称	timer_channel_output_fast_config
函数原型	void timer_channel_output_fast_config(uint32_t timer_periph, uint16_t channel, uint16_t ocfast);
功能描述	配置TIMERx通道输出比较快速功能
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<i>channel</i>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
<i>ocfast</i>	通道输出比较快速功能状态

<i>TIMER_OC_FAST_ENABLE</i>	通道输出比较快速功能使能
<i>TIMER_OC_FAST_DISABLE</i>	通道输出比较快速功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output fast function */
timer_channel_output_fast_config (TIMER0, TIMER_CH_0, TIMER_OC_FAST_ENABLE);
```

函数 **timer_channel_output_clear_config**

函数timer_channel_output_clear_config描述见下表：

表 3-675. 函数 **timer_channel_output_clear_config**

函数名称	timer_channel_output_clear_config
函数原型	void timer_channel_output_clear_config(uint32_t timer_periph, uint16_t channel, uint16_t occlear);
功能描述	配置TIMERx的通道输出比较清0功能
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
<i>channel</i>	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)

输入参数{in}	
occlear	通道比较输出清0功能状态
TIMER_OC_CLEAR_ENABLE	通道比较输出清0功能使能
TIMER_OC_CLEAR_DISABLE	通道比较输出清0功能禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output clear function */
timer_channel_output_clear_config (TIMER0, TIMER_CH_0,
TICKER_OC_CLEAR_ENABLE);
```

函数 **timer_channel_output_polarity_config**

函数**timer_channel_output_polarity_config**描述见下表：

表 3-676. 函数 **timer_channel_output_polarity_config**

函数名称	timer_channel_output_polarity_config
函数原型	void timer_channel_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocpolarity);
功能描述	通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)

<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
ocpolarity	通道输出极性
<i>TIMER_OC_POLARITY_HIGH</i>	通道输出极性高电平有效
<i>TIMER_OC_POLARITY_LOW</i>	通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 output polarity */
timer_channel_output_polarity_config (TIMER0, TIMER_CH_0,
TIMER_OC_Polarity_High);
```

函数 **timer_channel_complementary_output_polarity_config**

函数timer_channel_complementary_output_polarity_config描述见下表：

表 3-677. 函数 **timer_channel_complementary_output_polarity_config**

函数名称	timer_channel_complementary_output_polarity_config
函数原型	void timer_channel_complementary_output_polarity_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnpolarity);
功能描述	互补通道输出极性配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0,7)</i>	TIMER外设选择

输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0
<i>TIMER_CH_1</i>	通道1
<i>TIMER_CH_2</i>	通道2
输入参数{in}	
ocpolarity	互补通道输出极性
<i>TIMER_OCN_POLARITY_HIGH</i>	互补通道输出极性高电平有效
<i>TIMER_OCN_POLARITY_LOW</i>	互补通道输出极性低电平有效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output polarity */
timer_channel_complementary_output_polarity_config (TIMER0, TIMER_CH_0,
TICKER_OCN_POLARITY_HIGH);
```

函数 **timer_channel_output_state_config**

函数**timer_channel_output_state_config**描述见下表：

表 3-678. 函数 **timer_channel_output_state_config**

函数名称	timer_channel_output_state_config
函数原型	void timer_channel_output_state_config(uint32_t timer_periph, uint16_t channel, uint32_t state);
功能描述	配置通道状态
先决条件	-
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
state	通道状态
<i>TIMER_CCX_ENAB</i> <i>LE</i>	通道使能
<i>TIMER_CCX_DISA</i> <i>BLE</i>	通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 enable state */
timer_channel_output_state_config (TIMER0, TIMER_CH_0, TIMER_CCX_ENABLE);
```

函数 **timer_channel_complementary_output_state_config**

函数**timer_channel_complementary_output_state_config**描述见下表：

表 3-679. 函数 **timer_channel_complementary_output_state_config**

函数名称	timer_channel_complementary_output_state_config
函数原型	void timer_channel_complementary_output_state_config(uint32_t timer_periph, uint16_t channel, uint16_t ocnstate);
功能描述	配置互补通道输出状态
先决条件	-

被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0,7)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
TIMER_CH_2	通道2
输入参数{in}	
state	互补通道状态
TIMER_CCXN_ENA BLE	互补通道使能
TIMER_CCXN_DIS ABLE	互补通道禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 complementary output enable state */
timer_channel_complementary_output_state_config (TIMER0, TIMER_CH_0,
TICKER_CCXN_ENABLE);
```

函数 timer_channel_input_struct_para_init

函数timer_channel_input_struct_para_init描述见下表：

表 3-680. 函数 timer_channel_input_struct_para_init

函数名称	timer_channel_input_struct_para_init
函数原型	void timer_channel_input_struct_para_init(timer_ic_parameter_struct* icpara);
功能描述	将TIMER通道输入参数结构体中所有参数初始化为默认值

先决条件	-
被调用函数	-
输入参数{in}	
icpara	通道输入结构体, 详见 表 3-634. 结构体timer_ic_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* initialize TIMER channel input parameter struct with a default value */

timer_ic_parameter_struct timer_icinitpara;

timer_channel_input_struct_para_init(timer_icinitpara);
```

函数 timer_input_capture_config

函数timer_input_capture_config描述见下表:

表 3-681. 函数 timer_input_capture_config

函数名称	timer_input_capture_config
函数原型	void timer_input_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpara);
功能描述	配置TIMERx输入捕获参数
先决条件	-
被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)

<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
<i>icpara</i>	输入捕获结构体, 详见 表 3-634. 结构体timer_ic_parameter_struct 。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input capture parameter */
timer_ic_parameter_struct timer_icinitpara;

timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_capture_config(TIMER0, TIMER_CH_0, &timer_icinitpara);
```

函数 `timer_channel_input_capture_prescaler_config`

函数`timer_channel_input_capture_prescaler_config`描述见下表:

表 3-682. 函数 `timer_channel_input_capture_prescaler_config`

函数名称	timer_channel_input_capture_prescaler_config
函数原型	void timer_channel_input_capture_prescaler_config(uint32_t timer_periph, uint16_t channel, uint16_t prescaler);
功能描述	配置TIMERx通道输入捕获预分频值
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER外设
<i>TIMERx</i>	参考具体参数

输入参数{in}	
channel	待配置通道
<i>TIMER_CH_0</i>	通道0, TIMERx (x=0..4,7..13)
<i>TIMER_CH_1</i>	通道1, TIMERx (x=0..4,7,8,11)
<i>TIMER_CH_2</i>	通道2, TIMERx (x=0..4,7)
<i>TIMER_CH_3</i>	通道3, TIMERx (x=0..4,7)
输入参数{in}	
prescaler	通道输入捕获预分频值
<i>TIMER_IC_PSC_DI</i> V1	不分频
<i>TIMER_IC_PSC_DI</i> V2	2分频
<i>TIMER_IC_PSC_DI</i> V4	4分频
<i>TIMER_IC_PSC_DI</i> V8	8分频
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 channel 0 input capture prescaler value */  
  
timer_channel_input_capture_prescaler_config (TIMER0, TIMER_CH_0,  
TIMER_IC_PSC_DIV2);
```

函数 timer_channel_capture_value_register_read

函数timer channel capture value register read描述见下表：

表 3-683. 函数 timer channel capture value register read

功能描述	读取通道捕获值
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0, TIMERx (x=0..4,7..13)
TIMER_CH_1	通道1, TIMERx (x=0..4,7,8,11)
TIMER_CH_2	通道2, TIMERx (x=0..4,7)
TIMER_CH_3	通道3, TIMERx (x=0..4,7)
输出参数{out}	
-	-
返回值	
uint32_t	通道输入捕获值, (0x0000~0xFFFF)

例如：

```
/* read TIMER0 channel 0 capture compare register value */
uint32_t ch0_value = 0;
ch0_value = timer_channel_capture_value_register_read (TIMER0, TIMER_CH_0);
```

函数 **timer_input_pwm_capture_config**

函数timer_input_pwm_capture_config描述见下表：

表 3-684. 函数 **timer_input_pwm_capture_config**

函数名称	timer_input_pwm_capture_config
函数原型	void timer_input_pwm_capture_config(uint32_t timer_periph, uint16_t channel, timer_ic_parameter_struct* icpwm);
功能描述	配置TIMERx捕获PWM输入参数
先决条件	-

被调用函数	timer_channel_input_capture_prescaler_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8, 11)	TIMER外设选择
输入参数{in}	
channel	待配置通道
TIMER_CH_0	通道0
TIMER_CH_1	通道1
输入参数{in}	
icpwm	输入捕获结构体, 详见 表 3-634. 结构体timer_ic_parameter_struct
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 input pwm capture parameter */
timer_ic_parameter_struct timer_icinitpara;
timer_icinitpara.icpolarity = TIMER_IC_POLARITY_RISING;
timer_icinitpara.icselection = TIMER_IC_SELECTION_DIRECTTI;
timer_icinitpara.icprescaler = TIMER_IC_PSC_DIV1;
timer_icinitpara.icfilter = 0x0;
timer_input_pwm_capture_config (TIMER0, TIMER_CH_0, &timer_icinitpara);
```

函数 timer_hall_mode_config

函数timer_hall_mode_config描述见下表:

表 3-685. 函数 timer_hall_mode_config

函数名称	timer_hall_mode_config
函数原型	void timer_hall_mode_config(uint32_t timer_periph, uint8_t hallmode);

功能描述	配置TIMERx的HALL接口功能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
hallmode	HALL接口功能状态
TIMER_HALLINTERFACE_ENABLE	HALL接口使能
TIMER_HALLINTERFACE_DISABLE	HALL接口禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 hall sensor mode */
timer_hall_mode_config (TIMER0, TIMER_HALLINTERFACE_ENABLE);
```

函数 timer_input_trigger_source_select

函数timer_input_trigger_source_select描述见下表：

表 3-686. 函数 timer_input_trigger_source_select

函数名称	timer_input_trigger_source_select
函数原型	void timer_input_trigger_source_select(uint32_t timer_periph, uint32_t intrigger);
功能描述	TIMERx的输入触发源选择
先决条件	SMC[2:0] = 000
被调用函数	-
输入参数{in}	

timer_periph	TIMER外设
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER外设选择
输入参数{in}	
intrigger	待选择的触发源
<i>TIMER_SMCFG_T RGSEL_ITI0</i>	内部触发输入0(ITI0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_ITI1</i>	内部触发输入1(ITI1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_ITI2</i>	内部触发输入2(ITI2, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_ITI3</i>	内部触发输入3(ITI3, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_CI0F_ED</i>	CI0的边沿标志位 (CI0F_ED, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_CI0FE0</i>	滤波后的通道0输入 (CI0FE0, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_CI1FE1</i>	滤波后的通道1输入(CI1FE1, TIMERx(x=0..4,7,8,11))
<i>TIMER_SMCFG_T RGSEL_ETIFP</i>	滤波后的外部触发输入(ETIFP, TIMERx(x=0..4,7))
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 input trigger source */
timer_input_trigger_source_select (TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 timer_master_output_trigger_source_select

函数timer_master_output_trigger_source_select描述见下表：

表 3-687. 函数 timer_master_output_trigger_source_select

函数名称	timer_master_output_trigger_source_select
函数原型	void timer_master_output_trigger_source_select(uint32_t timer_periph, uint32_t outrigger);
功能描述	选择TIMERx主模式输出触发
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..7)	TIMER外设选择
输入参数{in}	
outrigger	主模式输出触发
TIMER_TRI_OUT_SRC_RESET	复位。TIMERx_SWEVG寄存器的UPG位被置1或从模式控制器产生复位触发一次TRGO脉冲，后一种情况下，TRGO上的信号相对实际的复位会有一个延迟。
TIMER_TRI_OUT_SRC_ENABLE	使能。此模式可用于同时启动多个定时器或控制在一段时间内使能从定时器。主模式控制器选择计数器使能信号作为触发输出TRGO。当CEN控制位被置1或者暂停模式下触发输入为高电平时，计数器使能信号被置1。在暂停模式下，计数器使能信号受控于触发输入，在触发输入和TRGO上会有一个延迟，除非选择了主/从模式。
TIMER_TRI_OUT_SRC_UPDATE	更新。主模式控制器选择更新事件作为TRGO。
TIMER_TRI_OUT_SRC_CC0	捕获/比较脉冲.通道0在发生一次捕获或一次比较成功时，主模式控制器产生一个TRGO脉冲
TIMER_TRI_OUT_SRC_O0CPRE	比较。在这种模式下主模式控制器选择O0CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O1CPRE	比较。在这种模式下主模式控制器选择O1CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O2CPRE	比较。在这种模式下主模式控制器选择O2CPRE信号被用于作为触发输出TRGO
TIMER_TRI_OUT_SRC_O3CPRE	比较。在这种模式下主模式控制器选择O3CPRE信号被用于作为触发输出TRGO
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* select TIMER0 master mode output trigger source */
timer_master_output_trigger_source_select (TIMER0, TIMER_TRI_OUT_SRC_RESET);
```

函数 timer_slave_mode_select

函数timer_slave_mode_select描述见下表：

表 3-688. 函数 timer_slave_mode_select

函数名称	timer_slave_mode_select
函数原型	void timer_slave_mode_select(uint32_t timer_periph, uint32_t slavemode);
功能描述	TIMERx从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8,11)	TIMER外设选择
输入参数{in}	
slavemode	从模式
TIMER_SLAVE_MODE_DISABLE	关闭从模式
TIMER_ENCODER_MODE0	编码器模式0
TIMER_ENCODER_MODE1	编码器模式1
TIMER_ENCODER_MODE2	编码器模式2
TIMER_SLAVE_MODE_RESTART	复位模式
TIMER_SLAVE_MODE_STOP	暂停模式

<i>DE_PAUSE</i>	
<i>TIMER_SLAVE_MODE_EVENT</i>	事件模式
<i>TIMER_SLAVE_MODE_EXTERNAL0</i>	外部时钟模式0
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* select TIMER0 slave mode */
timer_slave_mode_select(TIMER0, TIMER_ENCODER_MODE0);
```

函数 **timer_master_slave_mode_config**

函数timer_master_slave_mode_config描述见下表：

表 3-689. 函数 **timer_master_slave_mode_config**

函数名称	timer_master_slave_mode_config
函数原型	void timer_master_slave_mode_config(uint32_t timer_periph, uint8_t masterslave);
功能描述	TIMERx主从模式配置
先决条件	-
被调用函数	-
输入参数{in}	
<i>timer_periph</i>	TIMER外设
<i>TIMERx(x=0..4,7,8,11)</i>	TIMER外设选择
输入参数{in}	
<i>masterslave</i>	主从模式使能状态
<i>TIMER_MASTER_SLAVE_MODE_ENA_BLE</i>	主从模式使能

<i>TIMER_MASTER_SLAVE_MODE_DISABLE</i>	主从模式禁能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 master slave mode */
timer_master_slave_mode_config(TIMER0, TIMER_MASTER_SLAVE_MODE_ENABLE);
```

函数 **timer_external_trigger_config**

函数timer_external_trigger_config描述见下表：

表 3-690. 函数 **timer_external_trigger_config**

函数名称	timer_external_trigger_config
函数原型	void timer_external_trigger_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部触发输入
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输入参数{in}	
extprescaler	外部触发预分频
<i>TIMER_EXT_TRI_P_SC_OFF</i>	不分频
<i>TIMER_EXT_TRI_P_SC_DIV2</i>	2分频
<i>TIMER_EXT_TRI_P_SC_DIV4</i>	4分频

<i>TIMER_EXT_TRI_P</i> <i>SC_DIV8</i>	8分频
输入参数{in}	
expolarity	外部触发输入极性
<i>TIMER_ETP_FALLING</i>	低电平或者下降沿有效
<i>TIMER_ETP_RISING</i>	高电平或者上升沿有效
输入参数{in}	
extfilter	外部触发滤波控制 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 external trigger input */

timer_external_trigger_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 10);
```

函数 **timer_quadrature_decoder_mode_config**

函数**timer_quadrature_decoder_mode_config**描述见下表：

表 3-691. 函数 **timer_quadrature_decoder_mode_config**

函数名称	timer_quadrature_decoder_mode_config
函数原型	void timer_quadrature_decoder_mode_config(uint32_t timer_periph, uint32_t decomode,uint16_t ic0polarity, uint16_t ic1polarity);
功能描述	TIMERx配置为编码器模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择

输入参数{in}	
decomode	编码器模式
<i>TIMER_ENCODER_MODE0</i>	根据CI0FE0的电平，计数器在CI1FE1的边沿向上/下计数
<i>TIMER_ENCODER_MODE1</i>	根据CI1FE1的电平，计数器在CI0FE0的边沿向上/下计数
<i>TIMER_ENCODER_MODE2</i>	根据另一个信号的输入电平，计数器在CI0FE0和CI1FE1的边沿向上/下计数
输入参数{in}	
ic0polarity	IC0极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输入参数{in}	
ic1polarity	IC1极性
<i>TIMER_IC_POLARITY_RISING</i>	捕获上升边沿
<i>TIMER_IC_POLARITY_FALLING</i>	捕获下降边沿
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 quadrature decoder mode */
timer_quadrature_decoder_mode_config (TIMER0, TIMER_ENCODER_MODE0,
TICKER_IC_POLARITY_RISING, TICKER_IC_POLARITY_RISING);
```

函数 timer_internal_clock_config

函数timer_internal_clock_config描述见下表：

表 3-692. 函数 timer_internal_clock_config

函数名称	timer_internal_clock_config
函数原型	void timer_internal_clock_config(uint32_t timer_periph);
功能描述	TIMERx配置为内部时钟模式
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7,8, 11)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 internal clock mode */

timer_internal_clock_config (TIMER0);
```

函数 timer_internal_trigger_as_external_clock_config

函数timer_internal_trigger_as_external_clock_config描述见下表：

表 3-693. 函数 timer_internal_trigger_as_external_clock_config

函数名称	timer_internal_trigger_as_external_clock_config
函数原型	void timer_internal_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t intrigger);
功能描述	配置TIMERx的内部触发为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7,8,</i>	TIMER外设选择

11)	
输入参数{in}	
intrigger	被选择的内部触发源
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI0</i>	选择内部触发0 (ITI0)为时钟源
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI1</i>	选择内部触发1 (ITI1)为时钟源
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI2</i>	选择内部触发2 (ITI2)为时钟源
<i>TIMER_SMCFG_T</i> <i>RGSEL_ITI3</i>	选择内部触发3 (ITI3)为时钟源
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the internal trigger ITI0 as external clock input */
timer_internal_trigger_as_external_clock_config(TIMER0, TIMER_SMCFG_TRGSEL_ITI0);
```

函数 **timer_external_trigger_as_external_clock_config**

函数timer_external_trigger_as_external_clock_config描述见下表：

表 3-694. 函数 **timer_external_trigger_as_external_clock_config**

函数名称	timer_external_trigger_as_external_clock_config
函数原型	void timer_external_trigger_as_external_clock_config(uint32_t timer_periph, uint32_t exttrigger, uint16_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx的外部触发作为时钟源
先决条件	-
被调用函数	timer_input_trigger_source_select
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7,8,</i>	TIMER外设选择

11)	
输入参数{in}	
extrigger	外部触发源
<i>TIMER_SMCFG_T</i> <i>RGSEL_CI0F_ED</i>	CI0的边沿标志(CI0F_ED)
<i>TIMER_SMCFG_T</i> <i>RGSEL_CI0FE0</i>	滤波后的通道0输入(CI0FE0)
<i>TIMER_SMCFG_T</i> <i>RGSEL_CI1FE1</i>	滤波后的通道1输入(CI1FE1)
输入参数{in}	
expolarity	外部触发源极性
<i>TIMER_IC_POLARITY_RISING</i>	外部触发源高电平或者上升沿有效
<i>TIMER_IC_POLARITY_FALLING</i>	外部触发源低电平或者下降沿有效
输入参数{in}	
extfilter	滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external trigger CI0FE0 as external clock input */
timer_external_trigger_as_external_clock_config(TIMER0,
    TIMER_SMCFG_TRGSEL_CI0FE0, TIMER_IC_POLARITY_RISING, 0);
```

函数 timer_external_clock_mode0_config

函数timer_external_clock_mode0_config描述见下表：

表 3-695. 函数 timer_external_clock_mode0_config

函数名称	timer_external_clock_mode0_config
函数原型	void timer_external_clock_mode0_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);

功能描述	配置TIMERx外部时钟模式0，ETI作为时钟源
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7,8, 11)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_P SC_OFF	不分频
TIMER_EXT_TRI_P SC_DIV2	2分频
TIMER_EXT_TRI_P SC_DIV4	4分频
TIMER_EXT_TRI_P SC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLI NG	下降沿或者低电平有效
TIMER_ETP_RISIN G	上升沿或者高电平有效
输入参数{in}	
extfilter	ETI触发源滤波参数(0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode0 */
```

```
timer_external_clock_mode0_config(TIMER0, TIMER_EXT_TRI_PSC_DIV2,
TIMER_ETP_FALLING, 0);
```

函数 timer_external_clock_mode1_config

函数timer_external_clock_mode1_config描述见下表:

表 3-696. 函数 timer_external_clock_mode1_config

函数名称	timer_external_clock_mode1_config
函数原型	void timer_external_clock_mode1_config(uint32_t timer_periph, uint32_t extprescaler, uint32_t expolarity, uint32_t extfilter);
功能描述	配置TIMERx外部时钟模式1
先决条件	-
被调用函数	timer_external_trigger_config
输入参数{in}	
timer_periph	TIMER外设
TIMERx(x=0..4,7)	TIMER外设选择
输入参数{in}	
extprescaler	ETI触发源预分频值
TIMER_EXT_TRI_PSC_OFF	不分频
TIMER_EXT_TRI_PSC_DIV2	2分频
TIMER_EXT_TRI_PSC_DIV4	4分频
TIMER_EXT_TRI_PSC_DIV8	8分频
输入参数{in}	
expolarity	ETI触发源极性
TIMER_ETP_FALLING	下降沿或者低电平有效
TIMER_ETP_RISING	上升沿或者高电平有效
输入参数{in}	

extfilter	ETI触发源滤波参数 (0~15)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER0 the external clock mode1 */

timer_external_clock_mode1_config (TIMER0, TIMER_EXT_TRI_PSC_DIV2,
    TIMER_ETP_FALLING, 0);
```

函数 **timer_external_clock_mode1_disable**

函数**timer_external_clock_mode1_disable**描述见下表：

表 3-697. 函数 **timer_external_clock_mode1_disable**

函数名称	timer_external_clock_mode1_disable
函数原型	void timer_external_clock_mode1_disable(uint32_t timer_periph);
功能描述	TIMERx外部时钟模式1禁能
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7)</i>	TIMER外设选择
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable TIMER0 the external clock mode1 */

timer_external_clock_mode1_disable (TIMER0);
```

函数 timer_write_chxval_register_config

函数timer_write_chxval_register_config描述见下表:

表 3-698. 函数 timer_write_chxval_register_config

函数名称	timer_write_chxval_register_config
函数原型	void timer_write_chxval_register_config(uint32_t timer_periph, uint16_t ccsel);
功能描述	配置TIMERx写CHxVAL选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx(x=0..4,7..13)</i>	TIMER外设选择
输入参数{in}	
ccsel	写CHxVAL寄存器选择位
<i>TIMER_CHVSEL_DISABLE</i>	无影响
<i>TIMER_CHVSEL_ENABLE</i>	当写入捕获比较寄存器的值与寄存器当前值相等时，写入操作无效。
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure TIMER0 write CHxVAL register selection */

timer_write_chxval_register_config(TIMER0, TIMER_CHVSEL_ENABLE);
```

函数 timer_output_value_selection_config

函数timer_output_value_selection_config描述见下表:

表 3-699. 函数 timer_output_value_selection_config

函数名称	timer_output_value_selection_config
------	-------------------------------------

函数原型	void timer_output_value_selection_config(uint32_t timer_periph, uint16_t outsel);
功能描述	配置TIMER输出值选择位
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx (x=0,7)	TIMER外设选择
输入参数{in}	
ccsel	输出值选择位
TIMER_OUTSEL_D_ISABLE	无影响
TIMER_OUTSEL_E_NABLE	如果POEN位与IOS位均为0，则输出无效。
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure TIMER output value selection */

timer_output_value_selection_config(TIMER0, TIMER_OUTSEL_ENABLE);
```

函数 timer_interrupt_enable

函数timer_interrupt_enable描述见下表：

表 3-700. 函数 timer_interrupt_enable

函数名称	timer_interrupt_enable
函数原型	void timer_interrupt_enable(uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断使能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0..13)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_TRG	触发中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable the TIMER0 update interrupt */
timer_interrupt_enable (TIMER0, TIMER_INT_UP);
```

函数 timer_interrupt_disable

函数timer_interrupt_disable描述见下表:

表 3-701. 函数 timer_interrupt_disable

函数名称	timer_interrupt_disable
函数原型	void timer_interrupt_disable (uint32_t timer_periph, uint32_t interrupt);
功能描述	外设TIMERx中断禁能
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_UP	更新中断, TIMERx(x=0..13)
TIMER_INT_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
TIMER_INT_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_CH2	通道2比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_CH3	通道3比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_TRG	触发中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_BRK	中止中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable the TIMER0 update interrupt */
timer_interrupt_disable(TIMER0, TIMER_INT_UP);
```

函数 timer_interrupt_flag_get

函数timer_interrupt_flag_get描述见下表:

表 3-702. 函数 timer_interrupt_flag_get

函数名称	timer_interrupt_flag_get
函数原型	FlagStatus timer_interrupt_flag_get(uint32_t timer_periph, uint32_t interrupt);
功能描述	获取外设TIMERx中断标志
先决条件	-
被调用函数	-

输入参数{in}	
timer_periph	TIMER外设
<i>TIMERx</i>	参考具体参数
输入参数{in}	
interrupt	中断源
<i>TIMER_INT_FLAG_UP</i>	更新中断, TIMERx(x=0..13)
<i>TIMER_INT_FLAG_CH0</i>	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
<i>TIMER_INT_FLAG_CH1</i>	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_CH2</i>	通道2比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CH3</i>	通道3比较/捕获中断, TIMERx(x=0..4,7)
<i>TIMER_INT_FLAG_CMT</i>	换相更新中断, TIMERx(x=0,7)
<i>TIMER_INT_FLAG_TRG</i>	触发中断, TIMERx(x=0..4,7,8,11)
<i>TIMER_INT_FLAG_BRK</i>	中止中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update interrupt flag */
FlagStatus Flag_interrupt = RESET;

Flag_interrupt = timer_interrupt_flag_get (TIMER0, TIMER_INT_FLAG_UP);
```

函数 **timer_interrupt_flag_clear**

函数**timer_interrupt_flag_clear**描述见下表:

表 3-703. 函数 timer_interrupt_flag_clear

函数名称	timer_interrupt_flag_clear
函数原型	void timer_interrupt_flag_clear(uint32_t timer_periph, uint32_t interrupt);
功能描述	清除外设TIMERx的中断标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
interrupt	中断源
TIMER_INT_FLAG_UP	更新中断, TIMERx(x=0..13)
TIMER_INT_FLAG_CH0	通道0比较/捕获中断, TIMERx(x=0..4,7..13)
TIMER_INT_FLAG_CH1	通道1比较/捕获中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_CH2	通道2比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CH3	通道3比较/捕获中断, TIMERx(x=0..4,7)
TIMER_INT_FLAG_CMT	换相更新中断, TIMERx(x=0,7)
TIMER_INT_FLAG_TRG	触发中断, TIMERx(x=0..4,7,8,11)
TIMER_INT_FLAG_BRK	中止中断, TIMERx(x=0,7)
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear TIMER0 update interrupt flag */

timer_interrupt_flag_clear (TIMER0, TIMER_INT_FLAG_UP);
```

函数 timer_flag_get

函数timer_flag_get描述见下表：

表 3-704. 函数 timer_flag_get

函数名称	timer_flag_get
函数原型	FlagStatus timer_flag_get(uint32_t timer_periph, uint32_t flag);
功能描述	获取外设TIMERx的状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0..13)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0..4,7)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0..4,7)
TIMER_FLAG_CMT	通道换相更新标志, TIMERx(x=0,7)
TIMER_FLAG_TRG	触发标志, TIMERx(x=0,7,8,11)
TIMER_FLAG_BRK	中止标志位, TIMERx(x=0,7)
TIMER_FLAG_CH0_O	通道0捕获溢出标志, TIMERx(x=0..4,7..11)
TIMER_FLAG_CH1_O	通道1捕获溢出标志, TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	通道2捕获溢出标志, TIMERx(x=0..4,7)

O	
TIMER_FLAG_CH3_O	通道3捕获溢出标志, TIMERx(x=0..4,7)
输出参数{out}	
-	-
返回值	
FlagStatus	SET或者RESET

例如:

```
/* get TIMER0 update flags */

FlagStatus Flag_status = RESET;

Flag_status = timer_flag_get(TIMER0, TIMER_FLAG_UP);
```

函数 timer_flag_clear

函数timer_flag_clear描述见下表:

表 3-705. 函数 timer_flag_clear

函数名称	timer_flag_clear
函数原型	void timer_flag_clear(uint32_t timer_periph, uint32_t flag);
功能描述	清除外设TIMERx状态标志
先决条件	-
被调用函数	-
输入参数{in}	
timer_periph	TIMER外设
TIMERx	参考具体参数
输入参数{in}	
flag	状态标志
TIMER_FLAG_UP	更新标志, TIMERx(x=0..13)
TIMER_FLAG_CH0	通道0比较/捕获标志, TIMERx(x=0..4,7..13)
TIMER_FLAG_CH1	通道1比较/捕获标志, TIMERx(x=0..4,7,8,11)
TIMER_FLAG_CH2	通道2比较/捕获标志, TIMERx(x=0..4,7)
TIMER_FLAG_CH3	通道3比较/捕获标志, TIMERx(x=0..4,7)

<i>TIMER_FLAG_CMT</i>	通道换相更新标志, TIMERx(x=0,7)
<i>TIMER_FLAG_TRG</i>	触发标志, TIMERx(x=0,7,8,11)
<i>TIMER_FLAG_BRK</i>	中止标志位, TIMERx(x=0,7)
<i>TIMER_FLAG_CH0</i> O	通道0捕获溢出标志, TIMERx(x=0..4,7..11)
<i>TIMER_FLAG_CH1</i> O	通道1捕获溢出标志, TIMERx(x=0..4,7,8,11)
<i>TIMER_FLAG_CH2</i> O	通道2捕获溢出标志, TIMERx(x=0..4,7)
<i>TIMER_FLAG_CH3</i> O	通道3捕获溢出标志, TIMERx(x=0..4,7)
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* clear TIMER0 update flags */

timer_flag_clear (TIMER0, TIMER_FLAG_UP);
```

3.24. USART

通用同步异步收发器(USART)提供了一个灵活方便的串行数据交换接口, 章节[3.24.1](#)描述了USART的寄存器列表, 章节[3.24.2](#)对USART库函数进行说明。

3.24.1. 外设寄存器说明

USART寄存器列表如下表所示:

表 3-706. USART 寄存器

寄存器名称	寄存器描述
USART_STAT0	状态寄存器0
USART_DATA	数据寄存器

寄存器名称	寄存器描述
USART_BAUD	波特率寄存器
USART_CTL0	控制寄存器0
USART_CTL1	控制寄存器1
USART_CTL2	控制寄存器2
USART_GP	保护时间和预分频器寄存器
USART_CTL3	控制寄存器3
USART_RT	接收超时寄存器
USART_STAT1	状态寄存器1

3.24.2. 外设库函数说明

USART库函数列表如下表所示:

表 3-707. USART 库函数

库函数名称	库函数描述
uart_deinit	复位外设USART
uart_baudrate_set	配置USART波特率
uart_parity_config	配置USART奇偶校验
uart_word_length_set	配置USART字长
uart_stop_bit_set	配置USART停止位
uart_enable	使能USART
uart_disable	失能USART
uart_transmit_config	USART发送配置
uart_receive_config	USART接收配置
uart_data_first_config	配置数据传输时低位在前或高位在前
uart_invert_config	配置USART反转功能
uart_receiver_timeout_enable	使能USART接收超时
uart_receiver_timeout_disable	失能USART接收超时
uart_receiver_timeout_threshold_config	设置USART接收超时阈值

库函数名称	库函数描述
uart_data_transmit	USART发送数据功能
uart_data_receive	USART接收数据功能
uart_address_config	在地址掩码唤醒模式下配置USART地址
uart_mute_mode_enable	使能USART静默模式
uart_mute_mode_disable	失能USART静默模式
uart_mute_mode_wakeup_config	配置USART静默模式唤醒方式
uart_lin_mode_enable	使能USART LIN模式
uart_lin_mode_disable	失能USART LIN模式
uart_lin_break_detection_length_config	配置USART LIN模式中断帧长度
uart_send_break	配置USART发送断开帧
uart_halfduplex_enable	使能USART半双工模式
uart_halfduplex_disable	失能USART半双工模式
uart_synchronous_clock_enable	在USART同步通讯模式下使能CK引脚
uart_synchronous_clock_disable	在USART同步通讯模式下失能CK引脚
uart_synchronous_clock_config	配置USART同步通讯模式参数
uart_guard_time_config	在USART智能卡模式下配置保护时间值
uart_smartcard_mode_enable	使能USART智能卡模式
uart_smartcard_mode_disable	失能USART智能卡模式
uart_smartcard_mode_nack_enable	在USART智能卡模式下使能NACK
uart_smartcard_mode_nack_disable	在USART智能卡模式下失能NACK
uart_smartcard_autoretry_config	配置智能卡自动重试次数
uart_block_length_config	配置智能卡T=1的接收时块的长度
uart_irda_mode_enable	使能USART串行红外编解码功能模块
uart_irda_mode_disable	失能USART串行红外编解码功能模块
uart_prescaler_config	在USART IrDA低功耗模式下配置外设时钟分频系数
uart_irda_lowpower_config	配置USART IrDA低功耗模式
uart_hardware_flow_rts_config	配置USART RTS硬件控制流

库函数名称	库函数描述
uart_hardware_flow_cts_config	配置USART CTS硬件控制流
uart_dma_receive_config	配置USART DMA接收功能
uart_dma_transmit_config	配置USART DMA发送功能
uart_flag_get	获取USART状态寄存器标志位
uart_flag_clear	清除USART状态寄存器标志位
uart_interrupt_enable	使能USART中断
uart_interrupt_disable	失能USART中断
uart_interrupt_flag_get	获取USART中断标志位状态
uart_interrupt_flag_clear	清除USART中断标志位状态

函数 **uart_deinit**

函数uart_deinit描述见下表:

表 3-708. 函数 **uart_deinit**

函数名称	uart_deinit
函数原型	void usart_deinit(uint32_t usart_periph);
功能描述	复位外设USARTx/UARTx
先决条件	-
被调用函数	rcu_periph_reset_enable / rcu_periph_reset_disable
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset USART0 */
```

```
uart_deinit (USART0);
```

函数 **uart_baudrate_set**

函数uart_baudrate_set描述见下表:

表 3-709. 函数 **uart_baudrate_set**

函数名称	uart_baudrate_set
函数原型	void usart_baudrate_set(uint32_t usart_periph, uint32_t baudval);
功能描述	配置USART波特率
先决条件	-
被调用函数	rcu_clock_freq_get
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
baudval	波特率值
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure USART0 baud rate value */
uart_baudrate_set(USART0, 115200);
```

函数 **uart_parity_config**

函数uart_parity_config描述见下表:

表 3-710. 函数 **uart_parity_config**

函数名称	uart_parity_config
函数原型	void usart_parity_config(uint32_t usart_periph, uint32_t paritycfg);
功能描述	配置USART奇偶校验

先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
paritycfg	配置USART奇偶校验
<i>USART_PM_NONE</i>	无校验
<i>USART_PM_ODD</i>	奇校验
<i>USART_PM EVEN</i>	偶校验
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART parity */
uart_parity_config(USART0, USART_PM_EVEN);
```

函数 **uart_word_length_set**

函数uart_word_length_set描述见下表：

表 3-711. 函数 **uart_word_length_set**

函数名称	uart_word_length_set
函数原型	void usart_word_length_set(uint32_t usart_periph, uint32_t wlen);
功能描述	配置USART字长
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
wlen	配置USART字长
<i>USART_WL_8BIT</i>	8 bits
<i>USART_WL_9BIT</i>	9 bits
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 word length */
uart_word_length_set(USART0, USART_WL_9BIT);
```

函数 **uart_stop_bit_set**

函数uart_stop_bit_set描述见下表：

表 3-712. 函数 **uart_stop_bit_set**

函数名称	uart_stop_bit_set
函数原型	void uart_stop_bit_set(uint32_t usart_periph, uint32_t strlen);
功能描述	配置USART停止位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
strlen	配置USART停止位
<i>USART_STB_1BIT</i>	1 bit

<i>USART_STB_0_5BIT</i>	0.5 bit,该位对UARTx(x=3,4)无效
<i>USART_STB_2BIT</i>	2 bits
<i>USART_STB_1_5BIT</i>	1.5 bits,该位对UARTx(x=3,4)无效
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 stop bit length */
uart_stop_bit_set(USART0, USART_STB_1_5BIT);
```

函数 **uart_enable**

函数 **uart_enable** 描述见下表：

表 3-713. 函数 **uart_enable**

函数名称	uart_enable
函数原型	void uart_enable(uint32_t usart_periph);
功能描述	使能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 */

uart_enable(USART0);
```

函数 **uart_disable**

函数uart_disable描述见下表:

表 3-714. 函数 **uart_disable**

函数名称	uart_disable
函数原型	void usart_disable(uint32_t usart_periph);
功能描述	失能USART
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable USART0 */

uart_disable(USART0);
```

函数 **uart_transmit_config**

函数uart_transmit_config描述见下表:

表 3-715. 函数 **uart_transmit_config**

函数名称	uart_transmit_config
函数原型	void usart_transmit_config(uint32_t usart_periph, uint32_t txconfig);
功能描述	USART发送器配置
先决条件	-

被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
txconfig	使能/失能USART发送器
<i>USART_TRANSMIT_ENABLE</i>	使能USART发送
<i>USART_TRANSMIT_DISABLE</i>	失能USART发送
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 transmitter */
uart_transmit_config(USART0,USART_TRANSMIT_ENABLE);
```

函数 **usart_receive_config**

函数**usart_receive_config**描述见下表：

表 3-716. 函数 **usart_receive_config**

函数名称	usart_receive_config
函数原型	void usart_receive_config(uint32_t usart_periph, uint32_t rxconfig);
功能描述	USART接收器配置
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2

<i>UARTx</i>	x=3,4
输入参数{in}	
rxconfig	使能/失能USART接收器
USART_RECEIVE_ENABLE	使能USART接收
USART_RECEIVE_DISABLE	失能USART接收
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 receiver */
uart_receive_config(USART0, USART_RECEIVE_ENABLE);
```

函数 **uart_data_first_config**

函数**uart_data_first_config**描述见下表：

表 3-717. 函数 **uart_data_first_config**

函数名称	uart_data_first_config
函数原型	void usart_data_first_config(uint32_t usart_periph, uint32_t msbf);
功能描述	配置数据传输时低位在前或高位在前
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
msbf	数据传输时低位在前/高位在前
USART_MSBF_LSB	数据传输时低位在前

<i>USART_MSBF_MS_B</i>	数据传输时高位在前
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure LSB of data first */

uart_data_first_config(USART0, USART_MSBF_LSB);
```

函数 **uart_invert_config**

函数uart_invert_config描述见下表：

表 3-718. 函数 **uart_invert_config**

函数名称	uart_invert_config
函数原型	void uart_invert_config(uint32_t usart_periph, usart_invert_enum invertpara);
功能描述	配置USART反转功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
invertpara	参考枚举usart_invert_enum
<i>USART_DINV_ENA_BLE</i>	数据位电平反转
<i>USART_DINV_DIS_ABLE</i>	数据位电平不反转
<i>USART_TXPIN_EN_ABLE</i>	TX引脚电平反转
<i>USART_TXPIN_DIS_ABLE</i>	TX引脚电平不反转

<i>USART_RXPIN_ENABLE</i>	RX引脚电平反转
<i>USART_RXPIN_DISABLE</i>	RX引脚电平不反转
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART inversion */

uart_invert_config(USART0, USART_DINV_ENABLE);
```

函数 **uart_receiver_timeout_enable**

函数 **uart_receiver_timeout_enable** 描述见下表：

表 3-719. 函数 **uart_receiver_timeout_enable**

函数名称	uart_receiver_timeout_enable
函数原型	void usart_receiver_timeout_enable(uint32_t usart_periph);
功能描述	使能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable receiver timeout of USART */

uart_receiver_timeout_enable(USART0);
```

函数 `uart_receiver_timeout_disable`

函数`uart_receiver_timeout_disable`描述见下表:

表 3-720. 函数 `uart_receiver_timeout_disable`

函数名称	<code>uart_receiver_timeout_disable</code>
函数原型	<code>void usart_receiver_timeout_disable(uint32_t usart_periph);</code>
功能描述	失能USART接收超时
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx
<code>USARTx</code>	$x=0,1,2$
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* disable receiver timeout of USART */
uart_receiver_timeout_disable(USART0);
```

函数 `uart_receiver_timeout_threshold_config`

函数`uart_receiver_timeout_threshold_config`描述见下表:

表 3-721. 函数 `uart_receiver_timeout_threshold_config`

函数名称	<code>uart_receiver_timeout_threshold_config</code>
函数原型	<code>void usart_receiver_timeout_threshold_config(uint32_t usart_periph, uint32_t rtimeout);</code>
功能描述	设置USART接收超时阈值
先决条件	-
被调用函数	-
输入参数{in}	

usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
rtimeout	超时时间
<i>0-0xFFFFFFF</i>	超时时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* set the receiver timeout threshold of USART0 */
uart_receiver_timeout_threshold_config(USART0, 115200*3);
```

函数 **usart_data_transmit**

函数usart_data_transmit描述见下表：

表 3-722. 函数 **usart_data_transmit**

函数名称	usart_data_transmit
函数原型	void usart_data_transmit(uint32_t usart_periph, uint32_t data);
功能描述	USART发送数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
data	发送的数据
<i>0-0x1FF</i>	发送的数据
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* USART0 transmit data */

uart_data_transmit(USART0, 0xAA);
```

函数 **uart_data_receive**

函数uart_data_receive描述见下表：

表 3-723. 函数 **uart_data_receive**

函数名称	uart_data_receive
函数原型	uint16_t usart_data_receive(uint32_t usart_periph);
功能描述	USART接收数据功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
uint32_t	接收的数据 (0-0xFF)

例如：

```
/* USART0 receive data */

uint16_t temp;

temp = usart_data_receive(USART0);
```

函数 **uart_address_config**

函数uart_address_config描述见下表：

表 3-724. 函数 usart_address_config

函数名称	usart_address_config
函数原型	void usart_address_config(uint32_t usart_periph, uint8_t addr);
功能描述	在地址掩码唤醒模式下配置USART地址
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
addr	USART/UART地址
0-0xFF	USART/UART地址
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure address of the USART0 */
usart_address_config(USART0, 0x00);
```

函数 usart_mute_mode_enable

函数usart_mute_mode_enable描述见下表：

表 3-725. 函数 usart_mute_mode_enable

函数名称	usart_mute_mode_enable
函数原型	void usart_mute_mode_enable(uint32_t usart_periph);
功能描述	使能USART静默模式
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 receiver in mute mode */
uart_mute_mode_enable(USART0);
```

函数 **uart_mute_mode_disable**

函数uart_mute_mode_disable描述见下表：

表 3-726. 函数 **uart_mute_mode_disable**

函数名称	uart_mute_mode_disable
函数原型	void uart_mute_mode_disable (uint32_t usart_periph);
功能描述	失能USART静默模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* disable USART0 receiver in mute mode */

uart_mute_mode_disable(USART0);

```

函数 `uart_mute_mode_wakeup_config`

函数`uart_mute_mode_wakeup_config`描述见下表：

表 3-727. 函数 `uart_mute_mode_wakeup_config`

函数名称	<code>uart_mute_mode_wakeup_config</code>
函数原型	<code>void usart_mute_mode_wakeup_config(uint32_t usart_periph, uint32_t wmethod);</code>
功能描述	配置USART静默模式唤醒方式
先决条件	-
被调用函数	-
输入参数{in}	
<code>usart_periph</code>	外设USARTx/UARTx
<code>USARTx</code>	x=0,1,2
<code>UARTx</code>	x=3,4
输入参数{in}	
<code>wmethod</code>	两种方法用于进入或退出静默模式
<code>USART_WM_IDLE</code>	空闲线唤醒
<code>USART_WM_ADDR</code>	地址掩码唤醒
输出参数{out}	
-	-
返回值	
-	-

例如：

```

/* configure USART0 wakeup method in mute mode */

uart_mute_mode_wakeup_config(USART0, USART_WM_IDLE);

```

函数 `uart_lin_mode_enable`

函数`uart_lin_mode_enable`描述见下表：

表 3-728. 函数 usart_lin_mode_enable

函数名称	usart_lin_mode_enable
函数原型	void usart_lin_mode_enable(uint32_t usart_periph);
功能描述	使能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode enable */
usart_lin_mode_enable(USART0);
```

函数 usart_lin_mode_disable

函数usart_lin_mode_disable描述见下表：

表 3-729. 函数 usart_lin_mode_disable

函数名称	usart_lin_mode_disable
函数原型	void usart_lin_mode_disable(uint32_t usart_periph);
功能描述	失能USART LIN模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2

<i>UARTx</i>	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 LIN mode disable */

uart_lin_mode_disable(USART0);
```

函数 **uart_lin_break_decton_length_config**

函数 **uart_lin_break_decton_length_config** 描述见下表：

表 3-730. 函数 **uart_lin_break_decton_length_config**

函数名称	uart_lin_break_decton_length_config
函数原型	void usart_lin_break_decton_length_config(uint32_t usart_periph, uint32_t lrlen);
功能描述	配置USART LIN模式中断帧长度
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
lrlen	LIN模式中断帧长度
<i>USART_LBLEN_10_B</i>	断开帧长度为10 bits
<i>USART_LBLEN_11_B</i>	断开帧长度为11 bits
输出参数{out}	
-	-

返回值	
-	-

例如：

```
/* configure LIN break frame length */

uart_lin_break_dection_length_configUSART0, USART_LBLEN_10B);
```

函数 **uart_send_break**

函数uart_send_break描述见下表：

表 3-731. 函数 **uart_send_break**

函数名称	uart_send_break
函数原型	void usart_send_break(uint32_t usart_periph);
功能描述	配置USART发送断开帧
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 send break frame */

uart_send_breakUSART0);
```

函数 **uart_halfduplex_enable**

函数uart_halfduplex_enable描述见下表：

表 3-732. 函数 **uart_halfduplex_enable**

函数名称	uart_halfduplex_enable
------	------------------------

函数原型	void usart_halfduplex_enable(uint32_t usart_periph);
功能描述	使能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 half duplex mode*/
usart_halfduplex_enable(USART0);
```

函数 usart_halfduplex_disable

函数usart_halfduplex_disable描述见下表：

表 3-733. 函数 usart_halfduplex_disable

函数名称	usart_halfduplex_disable
函数原型	void usart_halfduplex_disable(uint32_t usart_periph);
功能描述	失能USART半双工模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* disable USART0 half duplex mode*/
uart_halfduplex_disable(USART0);
```

函数 **uart_synchronous_clock_enable**

函数uart_synchronous_clock_enable描述见下表：

表 3-734. 函数 **uart_synchronous_clock_enable**

函数名称	uart_synchronous_clock_enable
函数原型	void uart_synchronous_clock_enable(uint32_t usart_periph);
功能描述	在USART同步通讯模式下使能CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 CK pin in synchronous mode */
uart_synchronous_clock_enable(USART0);
```

函数 **uart_synchronous_clock_disable**

函数uart_synchronous_clock_disable描述见下表：

表 3-735. 函数 **uart_synchronous_clock_disable**

函数名称	uart_synchronous_clock_disable
------	--------------------------------

函数原型	void usart_synchronous_clock_disable(uint32_t usart_periph);
功能描述	在USART同步通讯模式下失能CK引脚
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 CK pin in synchronous mode */
usart_synchronous_clock_disable(USART0);
```

函数 usart_synchronous_clock_config

函数usart_synchronous_clock_config描述见下表：

表 3-736. 函数 usart_synchronous_clock_config

函数名称	usart_synchronous_clock_config
函数原型	void usart_synchronous_clock_config(uint32_t usart_periph, uint32_t clen, uint32_t cph, uint32_t cpl);
功能描述	配置USART同步通讯模式参数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
clen	CK信号长度

<i>USART_CLEN_NO NE</i>	8位数据帧中有7个CK脉冲，9位数据帧中有8个CK脉冲
<i>USART_CLEN_EN</i>	8位数据帧中有8个CK脉冲，9位数据帧中有9个CK脉冲
输入参数{in}	
cph	时钟相位
<i>USART_CPH_1CK</i>	在首个时钟边沿采样第一个数据
<i>USART_CPH_2CK</i>	在第二个时钟边沿采样第一个数据
输入参数{in}	
cpl	时钟极性
<i>USART_CPL_LOW</i>	CK引脚不对外发送时保持为低电平
<i>USART_CPL_HIGH</i>	CK引脚不对外发送时保持为高电平
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 synchronous mode parameters */

uart_synchronous_clock_config(USART0,USART_CLEN_EN,USART_CPH_2CK,
USART_CPL_HIGH);
```

函数 **usart_guard_time_config**

函数usart_guard_time_config描述见下表：

表 3-737. 函数 **usart_guard_time_config**

函数名称	usart_guard_time_config
函数原型	void usart_guard_time_config(uint32_t usart_periph,uint32_t gaut);
功能描述	在USART智能卡模式下配置保护时间值
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx

<i>USARTx</i>	x=0,1,2
输入参数{in}	
<i>guat</i>	保护时间值
0-0x000000FF	保护时间值
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 guard time value in smartcard mode */
uart_guard_time_config(USART0, 0x0000 0055);
```

函数 **uart_smartcard_mode_enable**

函数uart_smartcard_mode_enable描述见下表：

表 3-738. 函数 **uart_smartcard_mode_enable**

函数名称	uart_smartcard_mode_enable
函数原型	void uart_smartcard_mode_enable(uint32_t usart_periph);
功能描述	使能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
<i>usart_periph</i>	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 smartcard mode enable */
```

```
uart_smartcard_mode_enableUSART0);
```

函数 **uart_smartcard_mode_disable**

函数uart_smartcard_mode_disable描述见下表:

表 3-739. 函数 **uart_smartcard_mode_disable**

函数名称	uart_smartcard_mode_disable
函数原型	void usart_smartcard_mode_disable(uint32_t usart_periph);
功能描述	失能USART智能卡模式
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* USART0 smartcard mode disable */
uart_smartcard_mode_disableUSART0);
```

函数 **uart_smartcard_mode_nack_enable**

函数uart_smartcard_mode_nack_enable描述见下表:

表 3-740. 函数 **uart_smartcard_mode_nack_enable**

函数名称	uart_smartcard_mode_nack_enable
函数原型	void usart_smartcard_mode_nack_enable(uint32_t usart_periph);
功能描述	在USART智能卡模式下使能NACK
先决条件	-
被调用函数	-
输入参数{in}	

usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_enable(USART0);
```

函数 **uart_smartcard_mode_nack_disable**

函数uart_smartcard_mode_nack_disable描述见下表：

表 3-741. 函数 **uart_smartcard_mode_nack_disable**

函数名称	uart_smartcard_mode_nack_disable
函数原型	void uart_smartcard_mode_nack_disable(uint32_t usart_periph);
功能描述	在USART智能卡模式下失能NACK
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 NACK in smartcard mode */
```

```
uart_smartcard_mode_nack_disable(USART0);
```

函数 usart_smartcard_autoretry_config

函数usart_smartcard_autoretry_config描述见下表:

表 3-742. 函数 usart_smartcard_autoretry_config

函数名称	usart_smartcard_autoretry_config
函数原型	void usart_smartcard_autoretry_config(uint32_t usart_periph, uint32_t scrnum);
功能描述	配置智能卡自动重试次数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
scrnum	智能卡自动重试次数
0x0FFFFFFF	自动重试次数
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* configure smartcard auto-retry number */
usart_smartcard_autoretry_config (USART0, 0xFFFFFFFF);
```

函数 usart_block_length_config

函数usart_block_length_config描述见下表:

表 3-743. 函数 usart_block_length_config

函数名称	usart_block_length_config
函数原型	void usart_block_length_config(uint32_t usart_periph, uint32_t bl);
功能描述	配置智能卡T=1的接收时块的长度

先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	
bl	块长度
0-0xFFFFFFFF	块长度
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure block length in Smartcard T=1 reception */
uart_block_length_config(USART0, 0xFFFFFFFF);
```

函数 **uart_irda_mode_enable**

函数uart_irda_mode_enable描述见下表：

表 3-744. 函数 **uart_irda_mode_enable**

函数名称	uart_irda_mode_enable
函数原型	void uart_irda_mode_enable(uint32_t usart_periph);
功能描述	使能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
uart_periph	外设USARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	

-	-
返回值	
-	-

例如：

```
/* enable USART0 IrDA mode */
uart_irda_mode_enable(USART0);
```

函数 **uart_irda_mode_disable**

函数uart_irda_mode_disable描述见下表：

表 3-745. 函数 **uart_irda_mode_disable**

函数名称	uart_irda_mode_disable
函数原型	void usart_irda_mode_disable(uint32_t usart_periph);
功能描述	失能USART串行红外编解码功能模块
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 IrDA mode */
uart_irda_mode_disable(USART0);
```

函数 **uart_prescaler_config**

函数uart_prescaler_config描述见下表：

表 3-746. 函数 usart_prescaler_config

函数名称	usart_prescaler_config
函数原型	void usart_prescaler_config(uint32_t usart_periph, uint8_t psc);
功能描述	在USART IrDA低功耗模式下配置外设时钟分频系数
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
psc	时钟分频系数
0-0xFF	时钟分频系数
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure the USART0 peripheral clock prescaler in USART IrDA low-power mode */
usart_prescaler_config(USART0, 0x00);
```

函数 usart_irda_lowpower_config

函数usart_irda_lowpower_config描述见下表：

表 3-747. 函数 usart_irda_lowpower_config

函数名称	usart_irda_lowpower_config
函数原型	void usart_irda_lowpower_config(uint32_t usart_periph, uint32_t irlp);
功能描述	配置USART IrDA低功耗模式
先决条件	-
被调用函数	-

输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
irlp	IrDA低功耗模式或正常模式
USART_IRLP_LOW	低功耗模式
USART_IRLP_NORMAL	正常模式
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 IrDA low-power */
uart_irda_lowpower_config(USART0, USART_IRLP_LOW);
```

函数 **uart_hardware_flow_rts_config**

函数**uart_hardware_flow_rts_config**描述见下表：

表 3-748. 函数 **uart_hardware_flow_rts_config**

函数名称	uart_hardware_flow_rts_config
函数原型	void usart_hardware_flow_rts_config(uint32_t usart_periph, uint32_t rtsconfig);
功能描述	配置USART RTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
USARTx	x=0,1,2
输入参数{in}	

rtsconfig	使能/失能RTS
<i>USART_RTS_ENA BLE</i>	使能RTS
<i>USART_RTS_DISA BLE</i>	失能RTS
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control RTS */
uart_hardware_flow_cts_config(USART0, USART_RTS_ENABLE);
```

函数 **uart_hardware_flow_cts_config**

函数uart_hardware_flow_cts_config描述见下表：

表 3-749. 函数 **uart_hardware_flow_cts_config**

函数名称	uart_hardware_flow_cts_config
函数原型	void uart_hardware_flow_cts_config(uint32_t usart_periph, uint32_t ctsconfig);
功能描述	配置USART CTS硬件控制流
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx
<i>USARTx</i>	x=0,1,2
输入参数{in}	
ctsconfig	使能/失能CTS
<i>USART_CTS_ENA BLE</i>	使能CTS
<i>USART_CTS_DISA BLE</i>	失能CTS

输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* configure USART0 hardware flow control CTS */
uart_hardware_flow_cts_config(USART0, USART_CTS_ENABLE);
```

函数 usart_dma_receive_config

函数usart_dma_receive_config描述见下表：

表 3-750. 函数 usart_dma_receive_config

函数名称	usart_dma_receive_config
函数原型	void usart_dma_receive_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置USART DMA接收功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
dmacmd	使能/失能DMA接收功能
USART_DENR_ENABLE	使能DMA接收功能
USART_DENR_DISABLE	失能DMA接收功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for reception */

uart_dma_receive_config(USART0, USART_DENR_ENABLE);
```

函数 **uart_dma_transmit_config**

函数 **uart_dma_transmit_config** 描述见下表：

表 3-751. 函数 **uart_dma_transmit_config**

函数名称	uart_dma_transmit_config
函数原型	void usart_dma_transmit_config(uint32_t usart_periph, uint32_t dmacmd);
功能描述	配置 USART DMA发送功能
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
dmacmd	使能/失能DMA发送功能
<i>USART_DENT_ENABLE</i>	使能DMA发送功能
<i>USART_DENT_DISABLE</i>	失能DMA发送功能
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* USART0 DMA enable for transmission */

uart_dma_transmit_config(USART0, USART_DENT_ENABLE);
```

函数 usart_flag_get

函数usart_flag_get描述见下表：

表 3-752. 函数 usart_flag_get

函数名称	usart_flag_get
函数原型	FlagStatus usart_flag_get(uint32_t usart_periph, usart_flag_enum flag);
功能描述	获取USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
flag	USART标志位，参考枚举usart_flag_enum
USART_FLAG_CTS	CTS变化标志
USART_FLAG_LBD	LIN断开检测标志
USART_FLAG_TBE	发送数据缓冲区空标志
USART_FLAG_TC	发送完成标志
USART_FLAG_RB NE	读数据缓冲区非空标志
USART_FLAG_IDL EF	空闲线检测标志
USART_FLAG_OR ERR	溢出错误标志
USART_FLAG_NE RR	噪声错误标志
USART_FLAG_FER R	帧错误标志
USART_FLAG_PE RR	校验错误标志
USART_FLAG_BSY	忙状态标志

<i>USART_FLAG_EB</i>	块结束标志
<i>USART_FLAG_RT</i>	接收超时标志
输出参数{out}	
-	-
返回值	
FlagStatus	SET或RESET

例如：

```
/* get flag USART0 state */

FlagStatus status;

status = usart_flag_get(USART0,USART_FLAG_TBE);
```

函数 **usart_flag_clear**

函数usart_flag_clear描述见下表：

表 3-753. 函数 **usart_flag_clear**

函数名称	usart_flag_clear
函数原型	void usart_flag_clear(uint32_t usart_periph, usart_flag_enum flag);
功能描述	清除USART状态寄存器标志位
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
flag	USART标志位，参考枚举 usart_flag_enum
<i>USART_FLAG_CTS</i>	CTS变化标志
<i>USART_FLAG_LBD</i>	LIN断开检测标志
<i>USART_FLAG_TC</i>	发送完成
<i>USART_FLAG_RB</i>	读数据缓冲区非空

<i>NE</i>	
<i>USART_FLAG_EB</i>	块结束标志
<i>USART_FLAG_RT</i>	接收超时标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear USART0 flag */

uart_flag_clear(USART0,USART_FLAG_TC);
```

函数 **uart_interrupt_enable**

函数**uart_interrupt_enable**描述见下表：

表 3-754. 函数 **uart_interrupt_enable**

函数名称	uart_interrupt_enable
函数原型	void usart_interrupt_enable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	使能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
interrupt	USART中断
<i>USART_INT_PERR</i>	校验错误中断
<i>USART_INT_TBE</i>	发送缓冲区空中断
<i>USART_INT_TC</i>	发送完成中断
<i>USART_INT_RBNE</i>	读数据缓冲区非空中断和过载错误中断

<i>USART_INT_IDLE</i>	IDLE线检测中断
<i>USART_INT_LBD</i>	LIN断开信号检测中断
<i>USART_INT_ERR</i>	错误中断
<i>USART_INT_CTS</i>	CTS中断
<i>USART_INT_RT</i>	接收超时事件中断
<i>USART_INT_EB</i>	块结束事件中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* enable USART0 TBE interrupt */

uart_interrupt_enable(USART0, USART_INT_TBE);
```

函数 **uart_interrupt_disable**

函数uart_interrupt_disable描述见下表：

表 3-755. 函数 **uart_interrupt_disable**

函数名称	uart_interrupt_disable
函数原型	void uart_interrupt_disable(uint32_t usart_periph, usart_interrupt_enum interrupt);
功能描述	失能USART中断
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
interrupt	USART中断

<i>USART_INT_PERR</i>	校验错误中断
<i>USART_INT_TBE</i>	发送缓冲区空中断
<i>USART_INT_TC</i>	发送完成中断
<i>USART_INT_RBNE</i>	读数据缓冲区非空中断和过载错误中断
<i>USART_INT_IDLE</i>	IDLE线检测中断
<i>USART_INT_LBD</i>	LIN断开信号检测中断
<i>USART_INT_ERR</i>	错误中断
<i>USART_INT_CTS</i>	CTS中断
<i>USART_INT_RT</i>	接收超时事件中断
<i>USART_INT_EB</i>	块结束事件中断
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* disable USART0 TBE interrupt */
uart_interrupt_disable(USART0, USART_INT_TBE);
```

函数 **uart_interrupt_flag_get**

函数uart_interrupt_flag_get描述见下表：

表 3-756. 函数 **uart_interrupt_flag_get**

函数名称	uart_interrupt_flag_get
函数原型	FlagStatus uart_interrupt_flag_get(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	获取USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx

<i>USARTx</i>	x=0,1,2
<i>UARTx</i>	x=3,4
输入参数{in}	
<i>int_flag</i>	USART中断标志, 参考枚举uart_interrupt_flag_enum
<i>USART_INT_FLAG_PERR</i>	校验错误中断标志
<i>USART_INT_FLAG_TBE</i>	发送缓冲区空中断标志
<i>USART_INT_FLAG_TC</i>	发送完成中断标志
<i>USART_INT_FLAG_RBNE</i>	读数据缓冲区非空中断标志
<i>USART_INT_FLAG_RBNE_ORERR</i>	读数据缓冲区非空中断和溢出错误中断标志
<i>USART_INT_FLAG_IDLE</i>	IDLE线检测中断标志
<i>USART_INT_FLAG_LBD</i>	LIN断开检测中断标志
<i>USART_INT_FLAG_CTS</i>	CTS中断标志
<i>USART_INT_FLAG_ERR_ORERR</i>	过载错误中断标志
<i>USART_INT_FLAG_ERR_NERR</i>	噪声错误中断标志
<i>USART_INT_FLAG_ERR_FERR</i>	帧错误中断标志
<i>USART_INT_FLAG_EB</i>	块结束事件中断标志
<i>USART_INT_FLAG_RT</i>	超时事件中断标志
输出参数{out}	
-	-
返回值	

FlagStatus	SET或RESET
-------------------	-----------

例如：

```
/* get the USART0 interrupt flag status */

FlagStatus status;

status = usart_interrupt_flag_get(USART0, USART_INT_FLAG_RBNE);
```

函数 **usart_interrupt_flag_clear**

函数**usart_interrupt_flag_clear**描述见下表：

表 3-757. 函数 **usart_interrupt_flag_clear**

函数名称	usart_interrupt_flag_clear
函数原型	void usart_interrupt_flag_clear(uint32_t usart_periph, usart_interrupt_flag_enum int_flag);
功能描述	清除USART中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
usart_periph	外设USARTx/UARTx
USARTx	x=0,1,2
UARTx	x=3,4
输入参数{in}	
int_flag	USART中断标志
USART_INT_FLAG_CTS	CTS变化中断标志
USART_INT_FLAG_LBD	LIN断开检测中断标志
USART_INT_FLAG_TC	发送完成中断标志
USART_INT_FLAG_RBNE	读数据缓冲区非空中断标志
USART_INT_FLAG_EB	块结束事件中断标志

<code>USART_INT_FLAG_RT</code>	超时事件中断标志
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear the USART0 interrupt flag */

uart_interrupt_flag_clear(USART0, USART_INT_FLAG_RBNE);
```

3.25. WWDGT

窗口看门狗定时器(WWDGT)用来监测由软件故障导致的系统故障。章节[3.25.1](#)描述了WWDGT的寄存器列表，章节[3.25.2](#)对WWDGT库函数进行说明。

3.25.1. 外设寄存器说明

WWDGT寄存器列表如下表所示：

表 3-758. WWDGT 寄存器

寄存器名称	寄存器描述
<code>WWDGT_CTL</code>	控制寄存器
<code>WWDGT_CFG</code>	配置寄存器
<code>WWDGT_STAT</code>	状态寄存器

3.25.2. 外设库函数说明

WWDGT库函数列表如下表所示：

表 3-759. WWDGT 库函数

库函数名称	库函数说明
<code>wwdgt_deinit</code>	将WWDGT寄存器重设为缺省值
<code>wwdgt_enable</code>	使能WWDGT
<code>wwdgt_counter_update</code>	设置WWDGT计数器更新值
<code>wwdgt_config</code>	设置WWDGT计数器值、窗口值和预分频值

库函数名称	库函数说明
wwdgt_interrupt_enable	使能WWDGT提前唤醒中断
wwdgt_flag_get	检查WWDGT提前唤醒中断标志位是否置位
wwdgt_flag_clear	清除WWDGT提前唤醒中断标志位状态

函数 wwdgt_deinit

函数wwdgt_deinit描述见下表:

表 3-760. 函数 wwdgt_deinit

函数名称	wwdgt_deinit
函数原型	void wwdgt_deinit(void);
功能描述	将WWDGT寄存器重设为缺省值
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* reset the window watchdog timer configuration */
wwdgt_deinit();
```

函数 wwdgt_enable

函数wwdgt_enable描述见下表:

表 3-761. 函数 wwdgt_enable

函数名称	wwdgt_enable
函数原型	void wwdgt_enable (void);
功能描述	使能WWDGT
先决条件	-

被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* start the window watchdog timer counter */

wwdgt_enable ( );
```

函数 wwdgt_counter_update

函数wwdgt_counter_update描述见下表：

表 3-762. 函数 wwdgt_counter_update

函数名称	wwdgt_counter_update
函数原型	void wwdgt_counter_update(uint16_t counter_value);
功能描述	设置WWDGT计数器更新值
先决条件	-
被调用函数	-
输入参数{in}	
counter_value	0x00 - 0x7F
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* update WWDGT counter to 0x7F */

wwdgt_counter_update(127);
```

函数 wwdgt_config

函数wwdgt_config描述见下表:

表 3-763. 函数 wwdgt_config

函数名称	wwdgt_config
函数原型	void wwdgt_config(uint16_t counter, uint16_t window, uint32_t prescaler);
功能描述	设置WWDGT计数器值、窗口值和预分频值
先决条件	-
被调用函数	-
输入参数{in}	
counter	0x00 - 0x7F
输入参数{in}	
window	0x00 - 0x7F
输入参数{in}	
prescaler	WWDGT预分频值
WWDGT_CFG_PSC_DIV1	WWDGT计数器时钟为 (PCLK/4096) /1
WWDGT_CFG_PSC_DIV2	WWDGT计数器时钟为 (PCLK/4096) /2
WWDGT_CFG_PSC_DIV4	WWDGT计数器时钟为 (PCLK/4096) /4
WWDGT_CFG_PSC_DIV8	WWDGT计数器时钟为 (PCLK/4096) /8
输出参数{out}	
-	-
Return value	
-	-

例如:

```
/* confiure WWDGT counter value to 0x7F, window value to 0x50, prescaler divider value to
8 */

wwdgt_config(127, 80, WWDGT_CFG_PSC_DIV8);
```

函数 wwdgt_interrupt_enable

函数wwdgt_interrupt_enable描述见下表:

表 3-764. 函数 wwdgt_interrupt_enable

函数名称	wwdgt_interrupt_enable
函数原型	void wwdgt_interrupt_enable(void);
功能描述	使能WWDGT提前唤醒中断
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如:

```
/* enable early wakeup interrupt of WWDGT */
wwdgt_interrupt_enable();
```

函数 wwdgt_flag_get

函数wwdgt_flag_get描述见下表:

表 3-765. 函数 wwdgt_flag_get

函数名称	wwdgt_flag_get
函数原型	FlagStatus wwdgt_flag_get(void);
功能描述	检查WWDGT提前唤醒中断标志位是否置位
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	

-	-
返回值	
FlagStatus	SET or RESET

例如：

```
/* test if the counter value update has reached the 0x40 */

FlagStatus status;

status = wwdgt_flag_get ( );

if(status == RESET)

{
    ...

}

}else

{
    ...

}
}
```

函数 wwdgt_flag_clear

函数wwdgt_flag_clear描述见下表：

表 3-766. 函数 wwdgt_flag_clear

函数名称	wwdgt_flag_clear
函数原型	void wwdgt_flag_clear(void);
功能描述	清除WWDGT提前唤醒中断标志位状态
先决条件	-
被调用函数	-
输入参数{in}	
-	-
输出参数{out}	
-	-
返回值	
-	-

例如：

```
/* clear early wakeup interrupt state of WWDGT */  
wwdgt_flag_clear();
```

3.26. USBFS

USBFS 的固件库函数介绍请参考文档 **GD32F30x-Firmware-Library-USB User Manual_V1.0.**

4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	初稿发布	2018 年 12 月 26 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company under the intellectual property laws and treaties of the People's Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as its suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.